

ROBUST ANT COLONY BASED ROUTING ALGORITHM  
FOR MOBILE AD-HOC NETWORKS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Arush S. Sharma

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

August 2019

Purdue University

Indianapolis, Indiana

**THE PURDUE UNIVERSITY GRADUATE SCHOOL**  
**STATEMENT OF THESIS APPROVAL**

Dr. Dongsoo Kim, Chair

Department of Electrical and Computer Engineering

Dr. Brian King

Department of Electrical and Computer Engineering

Dr. Mohamed El-Sharkawy

Department of Electrical and Computer Engineering

**Approved by:**

Dr. Brian King

Head of Graduate Program

This work is dedicated to my parents who are the pillars of my life. Without their continued guidance and moral support, this work wouldn't have been possible. I will always be grateful to them.

## ACKNOWLEDGMENTS

I would like to thank my thesis advisor Dr. Dongsoo Kim and my thesis committee members Dr. Brian King and Dr. Mohamed El-Sharkawy for their continued guidance and advice during the course of this work. Many thanks to Mrs. Sherrie for her tireless efforts in reading this thesis. Special thanks to Dr. Kim for being a good mentor and for guiding me on the right path and Electrical and Computer Engineering department at Indiana University Purdue University Indianapolis (IUPUI) for their support.



## TABLE OF CONTENTS

	Page
LIST OF FIGURES . . . . .	vii
SYMBOLS . . . . .	ix
ABBREVIATIONS . . . . .	x
ABSTRACT . . . . .	xi
1 INTRODUCTION . . . . .	1
1.1 Design Challenges . . . . .	1
1.1.1 Background . . . . .	2
1.1.2 Overview . . . . .	3
2 LITERATURE REVIEW . . . . .	4
3 WORKING OF AODV ROUTING PROTOCOL . . . . .	9
4 MESSAGE FORMATS . . . . .	11
4.1 Foraging ant request format . . . . .	11
4.2 Reply ant message format . . . . .	12
4.3 Hello Messages . . . . .	13
5 WORKING OF TRADITIONAL ANT COLONY ROUTING PROTOCOL .	15
6 PROPOSED IDEA . . . . .	17
7 EXPERIMENTAL RESULTS . . . . .	21
7.1 Simulation Parameters . . . . .	21
7.2 Performance Metrics . . . . .	21
7.2.1 Simulation Results and Analysis . . . . .	23
7.2.2 Randomness Property of Ant Colony Routing Protocol in terms of Hop Count . . . . .	34
7.3 Importance of impact factors in Ant Colony Algorithm . . . . .	37
7.3.1 RSSI as an impact factor . . . . .	37

	Page
7.3.2 Hop Count as an impact factor . . . . .	38
7.3.3 Residual Energy as an impact factor . . . . .	39
8 COMPARATIVE ANALYSIS . . . . .	41
8.1 Background Traffic . . . . .	41
8.1.1 Throughput effect against background traffic . . . . .	41
8.1.2 Effect of Mean Delay against background traffic . . . . .	42
8.1.3 Effect of Packet Loss against background traffic . . . . .	43
8.2 Average Degree . . . . .	43
8.2.1 Throughput effect against average degree . . . . .	43
8.2.2 Effect of Mean Delay against Average Degree . . . . .	45
8.3 Energy Comparison . . . . .	45
8.3.1 Energy Depletion Series . . . . .	46
8.3.2 Comparison of Remaining Energy over different Time-stamps .	47
8.3.3 Comparison of standard deviation of residual energy . . . . .	51
8.3.4 Comparison of Average Remaining Energy . . . . .	52
8.4 Hop Count Comparison . . . . .	52
9 CONCLUSION AND FUTURE WORK . . . . .	55
REFERENCES . . . . .	56
A APPENDIX RSSI vs Distance Relationship . . . . .	58

## LIST OF FIGURES

Figure	Page
4.1 Format of foraging ant . . . . .	11
4.2 Reply Message Format . . . . .	12
4.3 Hello Message Format . . . . .	13
6.1 Goodness of RSSI against varying range of RSSI . . . . .	18
7.1 Placement of nodes . . . . .	22
7.2 Connectivity of the graph . . . . .	24
7.3 Throughput against background traffic . . . . .	25
7.4 Delay against background traffic . . . . .	26
7.5 Packet loss against background traffic . . . . .	27
7.6 Topology and connectivity of network. . . . .	28
7.7 Topology and connectivity of network continued. . . . .	29
7.8 Degree Distribution against varying transmission range . . . . .	30
7.9 Degree Distribution against varying transmission range (continued) . . . . .	31
7.10 Throughput vs Average Degree . . . . .	32
7.11 Delay vs Average Degree . . . . .	33
7.12 Packet Loss vs Average Degree . . . . .	34
7.13 Placement of nodes in a square grid . . . . .	35
7.14 Confidence interval in regards to hop count . . . . .	35
7.15 Hop Count Distribution against minimum hops . . . . .	36
7.16 Importance of RSSI Coefficient factor . . . . .	37
7.17 Importance of Hop Count factor . . . . .	38
7.18 Importance of Energy factor . . . . .	39
8.1 Throughput Comparison . . . . .	41
8.2 Mean Delay Comparison . . . . .	42

Figure	Page
8.3 Packet Loss Comparison . . . . .	43
8.4 Throughput Comparison against average degree . . . . .	44
8.5 Packet loss against average degree . . . . .	44
8.6 Mean Delay comparison against average degree . . . . .	45
8.7 Comparison of dead node series . . . . .	46
8.8 Color bar of remaining energy . . . . .	47
8.9 Remaining Energy over different time stamps . . . . .	48
8.10 Remaining Energy over different time stamps (continued) . . . . .	49
8.11 Ant Colony time stamps at 600 and 640 seconds . . . . .	50
8.12 Comparison of Standard Deviation of Residual Energy . . . . .	51
8.13 Comparison of average remaining energy . . . . .	52
8.14 Hop Comparison when Avg degree is 4.94 . . . . .	53
8.15 Hop Comparison when Avg degree is 5.94 . . . . .	53

## SYMBOLS

$\tau$	pheromone
$h$	hop count
$\alpha$	pheromone coefficient
$\beta$	hop count coefficient
$\delta$	energy coefficient
$\gamma$	goodness RSSI coefficient

## ABBREVIATIONS

ACO	Ant Colony Optimization
SI	Swarm Intelligence
AODV	Ad hoc On Demand Distance Vector
MANET	Mobile Ad hoc Networks
MAC	Media Access Control
WSN	Wireless Sensor Networks
RSSI	Received Signal Strength Indicator

## ABSTRACT

Sharma, Arush S. M.S.E.C.E., Purdue University, August 2019. Robust Ant Colony Based Routing Algorithm For Mobile Ad-hoc Networks. Major Professor: Dongsoo S. Kim Professor.

This thesis discusses about developing a routing protocol of mobile ad hoc networks in a bio inspired manner. Algorithms inspired by collective behaviour of social insect colonies, bird flocking, honey bee dancing, etc., promises to be capable of catering to the challenges faced by tiny wireless sensor networks. Challenges include but are not limited to low bandwidth, low memory, limited battery life, etc. This thesis proposes an energy efficient multi-path routing algorithm based on foraging nature of ant colonies and considers many other meta-heuristic factors to provide good robust paths from source node to destination node in a hope to overcome the challenges posed by resource constrained sensors.

# 1. INTRODUCTION

Wireless Sensor Networks (WSN) consist of a large number of nodes equipped with sensing capabilities; communication interfaces which has limited memory and energy resources. WSN nodes are statically deployed over large areas. However, they can also be mobile interacting with the environment. WSNs have wide spectrum of applications which includes environmental sensing, health care, traffic control, tracking of wild life animals, etc. Usually individual sensor nodes send their data towards base station node (commonly known as sink node). Intermediate nodes perform relaying of sensed data towards the destination.

## 1.1 Design Challenges

Following are the design challenges faced by WSNs Routing Protocols.

### a) Low Computational and memory requirements

Sensor nodes are equipped with a low end CPU and have limited memory. Therefore, it is mandatory that the routing algorithm has minimum overhead to make its execution feasible and effective.

### b) Self-organization

Wireless Sensor Network is expected to remain active for considerable period of time. Within that time period, few new nodes might be added to the network, while other nodes might die due to energy depletion or may become operational.



A routing protocol therefore should be robust to such dynamic and unpredictable events. It should be empowered with self-organizing properties to let the network function as an autonomous system.

### **c) Energy Efficiency**

Nodes are equipped with small non-rechargeable batteries. Therefore, the efficient battery usage of a sensor node is very important aspect to support the extended operational lifetime of network. The routing protocol is expected to forward the data packets across multiple paths, so that all nodes can deplete their battery source at a comparable rate. This results in load balance of network and increases the network lifetime.

### **d) Scalability**

In WSN applications, hundreds of nodes are generally deployed that have short communication ranges and high failure rates. Hence a routing protocol should be able to cope with the above challenges. It is imperative to design such routing protocols which can cater to the aforementioned challenges and are robust and adaptive. Nature provides us examples of mobile, independently working agents which seamlessly work together to perform tasks efficiently, for example, flight of migratory birds, ant colony optimization, etc. Nature inspired algorithms also known as Swarm Intelligence (SI) [1] are based on collective behavior of social insect colonies and other animal societies for solving different types of communication problems.

#### **1.1.1 Background**

Ant algorithms are special class of SI algorithms [2], which consist of population of simple agents (ants) which interact locally and with their environment. The foraging behavior of ant colony inspires Ant Colony algorithm. Initially, ants ran-

domly wander around the nest searching for food. When the food is found, they take it back to the colony and leave a trail of pheromones on the way. Other ants can sense the pheromone concentration and prefer to follow directions with higher pheromone density. Since shorter paths can be traversed faster, they will eventually outweigh the less optimal routes in terms of pheromone concentration. Additionally, pheromones evaporate over time, so ants are less likely to follow an older path which makes them search for newer paths simultaneously. In a case where an obstacle gets in their way, ants again initiates the route discovery process by randomly selecting the next hop until the ants converge on the paths with relatively higher concentration of pheromone.

### 1.1.2 Overview

The remainder of thesis is given as follows. At first general introduction and challenges are given which gives us the motivating factors to pursue research. Section 2 discusses about the related work done in the field of ant routing algorithms. Section 3 describes the working of well established AODV routing protocol. The description of AODV routing protocol is vital for the readers as ACO routing algorithm is compared against AODV in later section. Section 4 gives an overview of control packet (layer 3 control message) formats. Section 5 briefly describes about traditional ant colony routing algorithm. Section 6 talks about the novel ideas implemented in ant routing algorithm. Section 7 shows the robustness of proposed ant routing algorithm against various test scenarios. Section 8 does a comparison study against AODV routing protocol. Finally we conclude by giving a short summary and an outlook on future work.

## 2. LITERATURE REVIEW

AntNet proposed by Di Caro and Dorigo [3] is a routing technique which is applied for best-effort IP networks. Optimizing the performance of entire network is its main aim according to the principles of Ant Colony Optimization, AntNet is based on a greedy stochastic policy, where each node maintains a routing table and an additional table containing statistics about the traffic distribution over the network. The routing table maintains for each destination and for each next hop a measure of the goodness of using the next hop to forward data packets to destination. These goodness measures, called pheromone variables, are normalized on the stochastic policy. This algorithm uses forward ants and backward ants to update the routing table. The forward ants use heuristic based on the routing table to move between a pair of given nodes and are used to collect information about the traffic distribution over the network. The backward ant stochastically follows the path of forward ants in reverse direction. At each node, the backward ant updates the routing table and the additional table which contains traffic statistics of the network.

The energy-efficient ant-based routing algorithm (EEABR) is a routing protocol for WSNs and extends AntNet proposed by Tiago Camilo et. al., [4]. EEABR tries to minimize memory requirements as well as the overall energy consumption of the original AntNet algorithm. The ants retain information of only last two visited nodes because it takes into account the size of ant packet to update pheromone trail. In the typical ant-based algorithm, each ant carries the information of all the visited nodes. Then, in a network consisting of very large number of sensor nodes, the size of information would cause considerable energy to send ants through the network. Each node keeps the information of the received and sent ants in its memory.

Each memory record contains the previous node, the forward node, the ant identification, and a timeout value. The transmission probability considers the artificial pheromone value and the remaining energy of the possible next hop.

Ladder Diffusion Algorithm proposed by Ho et al. [5] addresses the energy consumption and routing problem in WSNs. The algorithm tries to reduce the energy consumption and processing time to build the routing table and avoid the route loop. In this algorithm, the sink node broadcasts the ladder creating packet with the grade value of one. The grade value of one means that the sensor node receiving this ladder creating packet transmits data to the sink node requires only one hop. Then sensor nodes increments the grade value of ladder creating packet and broadcast the modified ladder-creating packet. A grade value of two means that the sensor node receiving this ladder-creating packet sends data to the sink node requires two hop counts. And this step repeats until all the sensor nodes get the ladder-creating packet. The ladder diffusion algorithm assures that the direction of data transfer always occurs from a high grade value to a low grade value, which means each relay is forwarded to the sink node since each sensor node records the grade value of relay nodes in the ladder table. The path decision is based on the estimated energy consumption of path and the pheromone.

Energy-Aware Ant Routing in Wireless Multi-Hop Networks proposed by Michael Frey et. al., [6] provides new mechanisms for estimating the fitness of a path and energy information dissemination thus enabling to prolong the network lifetime. The network lifetime is the time span a network can fulfill its service. Traditional Ant Routing Algorithm considers the pheromone value in its probabilistic routing decision process. This approach favours shortest paths over non-shortest paths which is not suitable for energy constrained networks. EARA extends the ant routing algorithm with an energy heuristic for determining the nodes residual energy and scheme for estimating a paths energy. EARA algorithm considers the residual energy of a node as an additional heuristic.

Since the residual energy of a node changes over time, periodic Energy Ants are released for updating the energy values in the nodes routing table. Periodic Energy Ants are sent occasionally as it can be a costly operation in terms of consumed energy.

Ant Colony and Load Balancing Optimizations for AODV Routing Protocol proposed by Ahmed M. Abd Elmoniem et al., [7] discusses about improving the AODV routing protocol by taking the Ant Colony Optimization approach. Forward ant agents are sent as a part of route establishment request to find the route to destination. This route establishment phase is very much similar to Route Request (RREQ) phase of AODV routing protocol except for the fact that if the route to destination does not exist and there exist no neighbour, then the ant is broadcasted. Otherwise, if the active neighbour exists with highest pheromone, the forward ant is sent to that neighbour. In case of destination node receiving forward ant, backward ant is sent to the source node with a route to destination which comes under the part of Route establishment reply phase. The pheromone update policy is applied on the nodes receiving backward ants. Also it is applied differently depending upon whether the node is an source node or intermediate node or is destination node. Once the source node receives backward ant, the Data transmission phase begins. Each node receiving data packets forwards it to neighbor according to the pheromone values. Neighbor node having greater pheromone receives more data than those having less pheromone which leads to load balancing. If the route does not exist at all, a Route Error (RERR) packet is sent to the source node. If the routing table entry of the destination doesn't exist in source node, it deletes the route and again initiates the route discovery process.

Ant Colony Optimization for Routing and Load-Balancing: Survey and New Directions presented by Kwang Mong Sim et al., [8] provides comparison of the approaches for solving the convergence problem in ACO algorithms. When the network reaches its equilibrium state, the already discovered optimal path is given more preference over other paths by the ants which leads to many problems such as congestion, reduction of probability for selecting other paths, network failure, etc. In order to

mitigate this problem, some of the approaches include evaporation, aging, pheromone smoothing and limiting, privileged pheromone laying etc. Evaporation of pheromone is a technique to prevent the ants of favoring the older or stale paths which makes an ant to concurrently search for fresh paths. Aging refers to quantity of pheromone deposited by the ant. Older ant will deposit less pheromone compared to its young contemporary since they take more time in reaching destination. Limiting and Smoothing Pheromone refers to limiting the pheromone deposit by placing an upper bound which reduces preference of optimal paths over non-optimal paths In privileged pheromone laying, only certain ants are permitted to deposit extra pheromone. This makes the ant to converge to a solution by taking less time.

Ant-routing-algorithm (ARA) for mobile multi-hop ad-hoc networks- new features and results explored by Mesut Gunes et al., [9] is based on ant algorithms which makes it highly adaptive and efficient. The routing algorithm consists of three phases. Route Discovery Phase requires use of forward ant (FANT) and backward ant (BANT) control agents. FANT establishes the pheromone trail back to the source node. Similarly BANT establishes pheromone track back to the destination node. Node receiving FANT for the first time creates an entry in its routing table consisting of destination address which is the origin of FANT, next hop which is address of the previous node from which it received FANT and pheromone value which is computed based on the number of hops the FANT took to reach the node. The node forwards the FANT to its neighbors. Once the destination node receives FANT, it sends BANT back to the source node. Once the source nodes receives BANT from the destination node, the path is established and data packets can then be sent which comes under the Route Maintenance Phase. When data packets are relayed to destination by a node, it increases the pheromone value of the routing table entry. The last phase of ARA handles the routing failure caused by the mobility of node which are very common in MANETs. ARA assumes IEEE 802.11 on the MAC layer which enables routing algorithm to recognize the failure of route through a missing acknowledgement on the MAC layer. Node deactivates the link by setting the pheromone value

to 0. The node then searches for an alternative link in its routing table. If there exist a route to destination in its routing table, it sends the packet via this path. If there exist multiple en-tries in the routing table, the node will not send any data packets. Instead it informs the source node which has to initiate the route discovery process again.

### 3. WORKING OF AODV ROUTING PROTOCOL

Ad hoc On Demand Distance Vector (AODV) routing protocol [10] enables multi hop routing between source and destination node. It is reactive, i.e. on-demand routing protocol where the source node(s) doesn't initiate the route discovery process unless the route to destination is required. On the other hand routing protocols which come under the category of proactive routing continues to maintain routes between source and destination even when the route to destination is not required. AODV routing protocol consists of two phases: i) route discovery and ii) route maintenance.

When a source node wishes to communicate with some destination node, it first seeks for a route in its routing table. If the route exists, then the communication between two starts immediately. If not, then route discovery process is initiated. The route discovery process consists of broadcasting a route request (RREQ) message. If one of the intermediate node receiving RREQ message has a valid route to destination, it replies back with a route reply (RREP) message. Otherwise RREQ message is broadcasted by intermediate nodes until it reaches the destination node. The intermediate node while handling RREQ message increments the hop count value in RREQ packet by one. This accounts for hop count required to reach the source node. Additionally intermediate nodes create a routing table entry which contains address of source node, total number of hops required to reach the source and the next hop's address which is IPV4 address of neighbor node from whom it received the message. Each routing table entry is associated with lifetime, i.e. if the route entry is not used within the lifetime, it will be deleted from routing table. In this way the destination node becomes aware of source node and generates RREP message.

The RREP is unicast to the next hop towards the originator of RREQ message which is indicated by routing table entry for the source node. Intermediate nodes receiving RREP packet increments the hop count field in routing table by one. When



RREP is received by source node, the hop count field represents the total distance, in terms of hops, to reach destination node. This completes the route discovery phase.

The second phase of the protocol is route maintenance. It is performed by source node when the destination or an intermediate node moves. A route error message (RERR) is sent to the source node. Intermediate nodes receiving RERR message update their routing table entry for destination by setting the hop count field to infinity. The source node receiving RERR initiates the route discovery process again.

## 4. MESSAGE FORMATS

### 4.1 Foraging ant request format

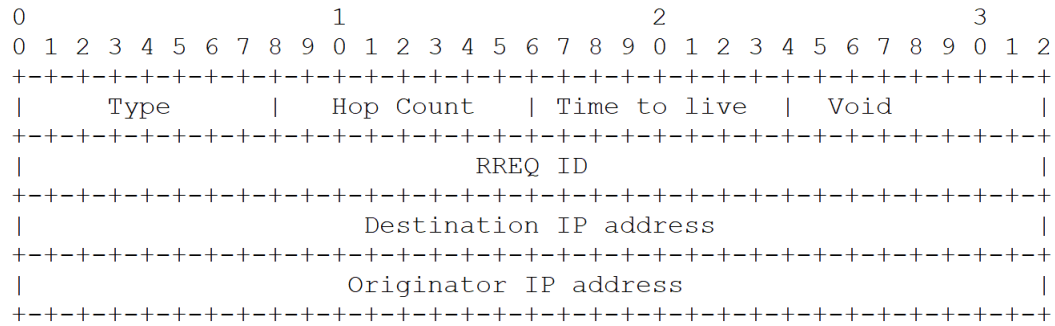


Fig. 4.1. Format of foraging ant

The format of the foraging ant control message is illustrated above and contains the following fields:

Type	2
Hop Count	Number of hop counts between the node receiving the foraging ant and the originator node which initiated the request
Time to Live	Counter mechanism to limit the lifetime of control packet

RREQ ID	A sequence number identifying the foraging ant when taken in consideration with the originating node's IP address.
Destination IP Address	The IP Address of destination node
Originator IP Address	The IP Address of source node which initiated the route discovery process

## 4.2 Reply ant message format

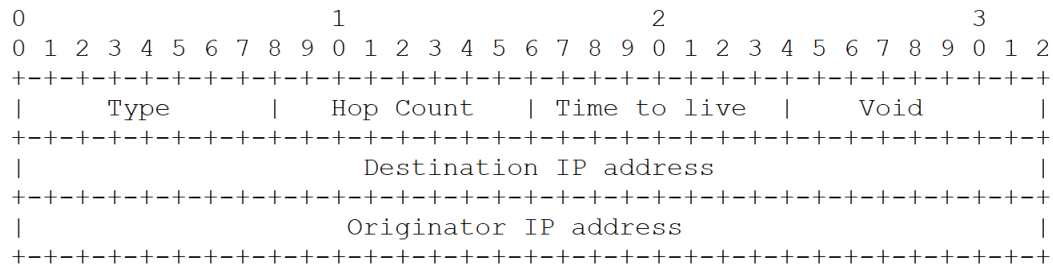


Fig. 4.2. Reply Message Format

The format of the reply ant control message is shown above, and contains the following fields:

Type	3
Hop Count	Number of hops between the node receiving reply message and destination node which initiated the reply message
Time to Live	Counter mechanism to limit the life-time of control packet

Destination IP Address	The IP Address of destination node
Originator IP Address	The IP Address of source node which initiated the route discovery process

### 4.3 Hello Messages

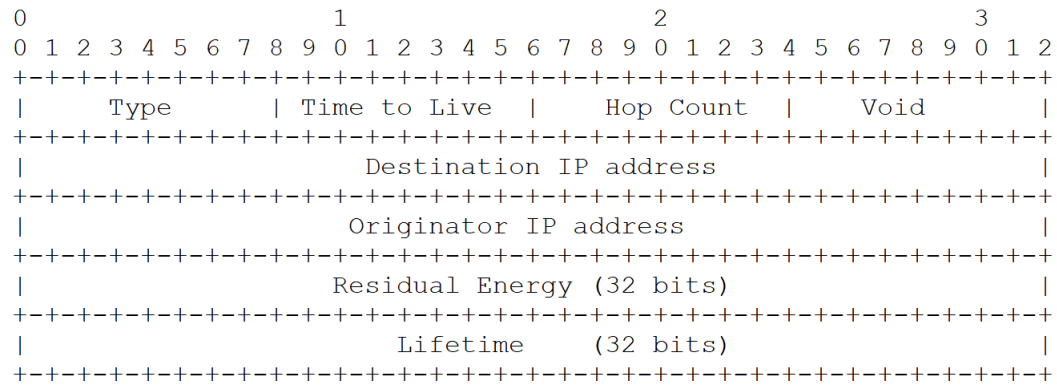


Fig. 4.3. Hello Message Format

The format of hello control message is shown below and contains the following fields:

Type	1
Time to Live	Counter mechanism to limit the life-time of control packet. It is initialized to 1.
Hop Count	0

Destination IP Address	The node's IP Address
Originator IP Address	Neighbor's IP Address
Residual Energy	Remaining energy of node from whom it received the hello message
Lifetime	Time till which the link between neighbor node is considered to be active. If it doesn't receive Hello message within that time period, the node assumes that its active link to neighbor is lost

## 5. WORKING OF TRADITIONAL ANT COLONY ROUTING PROTOCOL

The basic ACO takes a reactive probabilistic approach of finding good robust paths between source and destination. The algorithm follows: At regular intervals, a foraging ant is launched with a mission to find a path to destination. This establishes back-ward pheromone trail from destination to source. When an intermediate node receives a foraging ant for the first time, it creates an entry in its routing table. The entry consists of destination address which is the source address of the foraging ant, next hop which implies the node from which it received the packet, and pheromone value. The intermediate node increments the pheromone value as given by following equation:

$$\tau_n^d(t) = \tau(t - \delta t) + \delta\tau \quad (5.1)$$

where  $\tau_n^d(t)$  is the current pheromone value present at the routing table of node  $n$  to reach destination node  $d$ ,  $\delta t$  is the time duration for which it received the last pheromone,  $\delta\tau$  is the incremental pheromone. The foraging ant probabilistically selects the next hop to reach destination using the formula given below.

$$P_{n,d} = \frac{(\tau_{n,d})^\beta}{\sum_{j \in N} (\tau_{j,d})^\beta} \quad (5.2)$$

where  $P_{n,d}$  is a probability to select neighbor  $n$  as a next hop towards destination  $d$ ,  $\tau_{n,d}$  is a pheromone value at neighbor  $n$  to reach destination  $d$ ,  $N$  is the set of neighbors and  $\beta$  is a pheromone coefficient constant.

Duplicate foraging ants are then removed by identifying their unique sequence ID. Once a forward ant reaches its destination, it initiates uni-cast forwarding of backward ants which the destination will send to the source node. The backward ant establishes the pheromone trail from source to destination. After calculating the selection probabilities, the node will forward the data packets to that neighbor node

which has been selected based on the distribution of Equation 5.2. The data packet is sent to the selected relayed node and is further relayed towards the destination node. The selected relay nodes increments their pheromone value by a specific amount. Like their natural counterpart, artificial pheromones decay over time. The evaporation process provides a negative feedback in the system which helps ant avoid the stale paths in the network. The evaporation of pheromones takes place constantly by equation given below.

$$\tau_n^d(t) = \tau_n^d(\delta t) * e^{-(t-\delta t)\rho} \quad (5.3)$$

where  $\rho$  is constant called evaporation rate of pheromone.

The procedure finishes once the data packet reaches the destination node.

Nodes maintain the neighbor entity in its neighbor table by sending Hello Packet periodically to each other. Hello packet sending interval can be different according to different mobility scenarios. If a node doesn't receive Hello packet from a neighbor for a certain period of time, it then deletes the neighbor information from its neighbor table.

## 6. PROPOSED IDEA

This section discusses about various heuristic factors which are considered in the proposed routing algorithm which makes it novel compared to already existing routing algorithms for MANETs.

**Pheromone and Repellent Pheromone:** Ants in nature while travelling from their nest to food source make the routing decision when they reach intersection, i.e., when more than one path is available for their next hop. In such scenario, the probability to choose that path is more which has relatively higher concentration of pheromone compared to other available paths. After the robust path is established, the ants continue to use that path until they encounter some obstacle, for example, placing a stone or pouring water on the path recently formed by ants. In such case, ants no longer use that path and instead begin exploring new paths. Pheromones with repellent property are deposited by ants so that their contemporaries no longer use the earlier efficient path. This property if incorporated in algorithm would make delay less in the network as the nodes would be aware of failed paths in the network.

**RSSI:** In WSN, sensor nodes are aware of the proximity of their neighbors through RSSI. If the scenario is considered where source node and destination node are placed very far apart such that the destination node barely comes under the transmission range of source node, both the nodes will receive packets with very low RSSI as power of received signal decreases with increase in distance. As a result the chances of packet drops are very high. But due to less number of hops between source and destination node, the destination node will experience less delay. Whereas if source node and destination node are connected such that there exist neighboring nodes through which the packets can be transferred in a multi-hop fashion, then the nodes will receive packets with very high value of RSSI due to close vicinity with each other.



Nevertheless, the delay experienced by the packet will be more as the packet would have travelled with more number of hops from source to destination.

Therefore, it is understood that extreme values of RSSI is not appreciated for our pro-posed system. With the logic of RSSI explained above, it is vital that the goodness of RSSI closely follows the Gaussian distribution [11] as shown in the figure below.

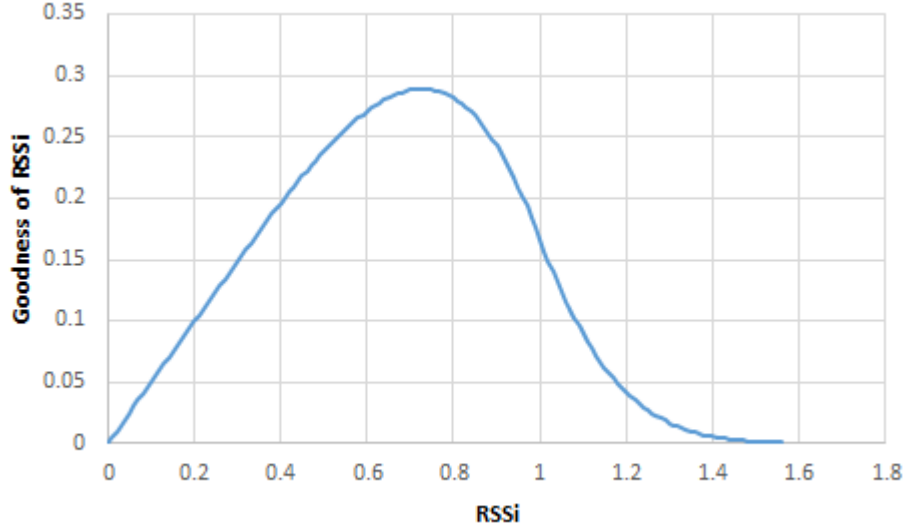


Fig. 6.1. Goodness of RSSI against varying range of RSSI

With reference to [12], if two nodes (source and destination) are placed such that the distance between them is less than the transmission radius  $R$ , the total expected hop count from source node to destination node is given by

$$\frac{d}{x} \left[ \frac{1}{p(x)[(1 - (1 - p(x))^u)]} + \frac{u}{[1 - (1 - p(x))^u]} \right] \quad (6.1)$$

where  $x$  is the distance between two consecutive nodes.  $p(x)$  is the probability of receiving packet and  $u$  is a constant and is selected as 1.  $p(x)$  is dependent on several measurements such as signal strengths, delay, etc. It is approximated by the following equation:

$$P(x) = \begin{cases} 1 - \left(\frac{x}{R}\right)^{2\beta} / 2, & \text{for } x < R \\ ((2R - x)/R)^{2\beta} / 2, & \text{for } x \geq R \end{cases} \quad (6.2)$$

Where  $\beta$  is constant and is selected as 2. In the experiment,  $R$  is selected such that the packet delivery ratio at destination node is 80%.

In order to make equation 6.1 independent of particular distance  $d$ , it is optimized which is given as follows:

$$h(x, \mu, \beta, R) = \frac{R}{x} \left[ \frac{1}{[p(x)(1 - (1 - p(x))^u)]} + \frac{u}{[(1 - (1 - p(x))^u)]} \right] \quad (6.3)$$

By plugging in the value of  $u$  in equation 6.3, it is simplified by following equation

$$h(x) = \frac{R(1 + p(x))}{xp^2(x)} \quad (6.4)$$

Taking the inverse of above equation, we claim that the Goodness of RSSI [13] is achieved which follows the graph in Figure 6.1. Hence

$$f(S) = \frac{1}{h(x)} = \frac{xp^2(x)}{R(1 + p(x))} \quad (6.5)$$

Where  $f(S)$  is known as Goodness RSSI. After the goodness of RSSI is calculated, the probabilistic formulae and goodness RSSI follows linear relationship. Each node then maintains a neighbor table where RSSI records are maintained against every neighbor. Also it is widely known that RSSI fluctuates too often even when nodes are static [14], the exponential weighted moving average (EWMA) approach chosen by us helps in smoothening the RSSI value.

**Residual Energy:** If the nodes among the discovered robust path are going to be used extensively for data packet transmission, their battery will deplete faster compared to nodes on non-efficient paths. This will result in creation of void nodes in the network which may lead to network partition. Inclusion of residual energy of node [15] in the routing decision will help in exploring paths other than already discovered robust paths. This technique will improve the lifetime of the network.

Hops: The routing tables at each node gets modified by information from the incoming packets. Through the backward learning the node learns the identity of source as well as destination node and also the total hops required to reach them. If the previous value of hop count stored in the routing table of node is better than the current one then nothing is done but if the current value of hop count is better than the previous one, then the value is updated for future use.

The proposed routing algorithm is an extended version of traditional ant colony based routing algorithm in which the main objective is to maximize the network life-time. Traditional ant colony routing algorithm considers the pheromone value alone in its probabilistic routing decision process which is not favorable for energy constrained networks. With additional heuristics discussed above, the extended probabilistic formula is defined as:

$$P_{n,d} = \frac{(\tau_{n,d})^\alpha (h_{n,d})^{-\beta} (E_n)^\delta (f(S)_n)^\gamma}{\sum_{j \in N} (\tau_{j,d})^\alpha (h_{j,d})^{-\beta} (E_{j,d})^\delta (f(S)_{j,d})^\gamma} \quad (6.6)$$

where  $n$  is the next hop selected by an ant to reach destination  $d$ ,  $\tau_{n,d}$  is a pheromone value from neighbor  $n$  to reach destination  $d$ .  $h$  is the number of hops taken by an ant to reach destination node  $d$ .  $N$  is the set of neighbors of node.  $E$  is the remaining energy in the node,  $f(S)$  is the goodness RSSI which follows Gaussian distribution as shown in Figure 6.1.  $\alpha, \beta, \delta$  and  $\gamma$  are the factors to adjust the relative importance of pheromone concentration, hops, residual energy and goodness RSSI respectively.

As discussed in Section 2, nodes receiving the ants update the pheromone value in their routing table by depositing a constant value of pheromone in their routing table which acts as a positive feedback. As a result, an impulsive response is observed with regards to pheromone whenever a node receives an ant. Similar to the biological ants, the pheromone value is a function of time which means pheromone value decreases exponentially as the time progresses which makes it volatile.

## 7. EXPERIMENTAL RESULTS

### 7.1 Simulation Parameters

Ant Colony routing algorithm is implemented on open source network simulator (ns-3) [16]. Nodes are initially laid out in regular hexagonal structure to account for hexagon shape used for radio coverage in cellular communication system. The deployment of nodes are then made random by adding Gaussian Random Variable with variance (99) to its x and y coordinates. This deployment of nodes can be named as hexagonal randomized placement. As shown in Figure 7.1, the source and destination nodes are placed along the diagonal so that they are far apart. By default 802.11b is used as underlying MAC protocol (Layer-2). UDP Echo is used as the application layer protocol. Simulation time for each experiment is set to at-least 500 seconds. Performance of routing protocol is carried out by measuring the performance metrics as described in the next section.

### 7.2 Performance Metrics

#### a) Throughput

It is the rate at which the data packets are delivered successfully by the network to destination node from source node. Also known as goodput, it is represented by bits/bytes per second. The throughput is affected by various factors such as background traffic/noise, bandwidth of physical medium, processing power, end user medium, etc. In any communication based network, higher throughput is always desired.

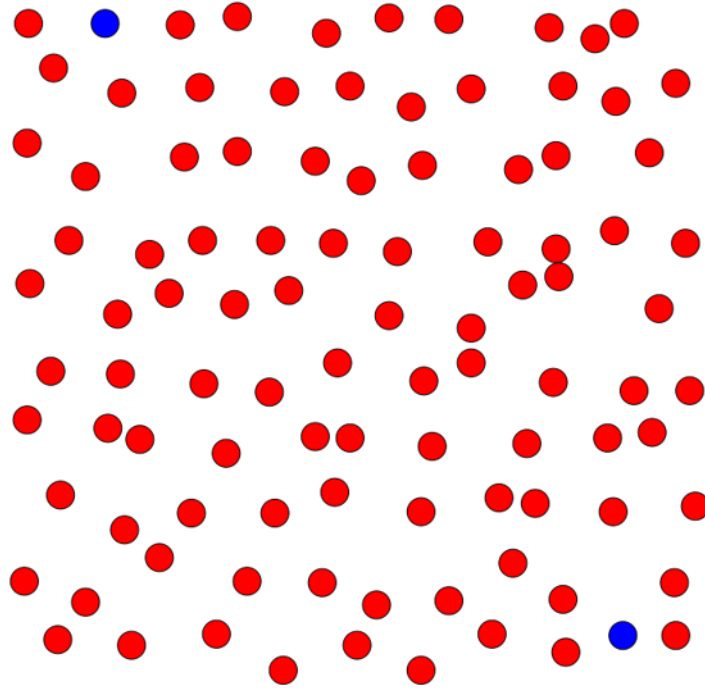


Fig. 7.1. Placement of nodes

### b) Mean Delay

Mean Delay is the time consumed by the data packet to reach the destination node from source node. With reference to [17] mean delay is calculated by taking the ratio of delay sum to the total number of received packets at the destination node.

### c) Packet Loss

Packet Loss results when one or more data packets fail to reach the destination node due to various reasons such as dropping of packets, error in data transmission, network congestion due to overwhelming loads.

#### **d) Network Lifetime**

It is defined as the time span a network can fulfill its service whereby source node and destination node communicate with each other by exchanging data packets and other control packets. Also in various literature's, it is the time at which the first node in the network becomes dead.

#### **e) Partition Time**

Partition Time is defined as the time beyond which communication between source and destination no longer takes place. This happens when the network becomes disconnected due to energy depleted nodes.

#### **f) Mean Hop Count**

Mean Hop Count is the average of hops taken by the data packet to reach the destination node from source node.

### **7.2.1 Simulation Results and Analysis**

The performance of routing protocol is analyzed under different test scenarios and topologies to account for the robustness.

#### **a) Background Traffic**

The logic of introducing background traffic in simulation is to mimic real life network scenarios where there's a disturbance or white noise along with regular traffic flow. At every regular time interval (50 seconds) a pair of random nodes are chosen as background source and destination nodes which does the job of background traffic flow. At the end of simulation, there are 10 pairs of nodes which contribute to the background traffic. During simulation run-time, every node from MAC (L-2) layer is

able to keep track of how many packets it has sent, how many packets it has received and how many packets it has dropped during transmission and reception of packet. These results are analyzed at that time when the pair of nodes are chosen randomly for background traffic flow and then the average of resultant is calculated. Hence these attributes are vital for background traffic calculation.

The topology of graph is shown in Figure 7.2. The minimum average degree of node is 4.94 for the graph to be connected. AML2FIG software [18] is used to show the topology of the network.

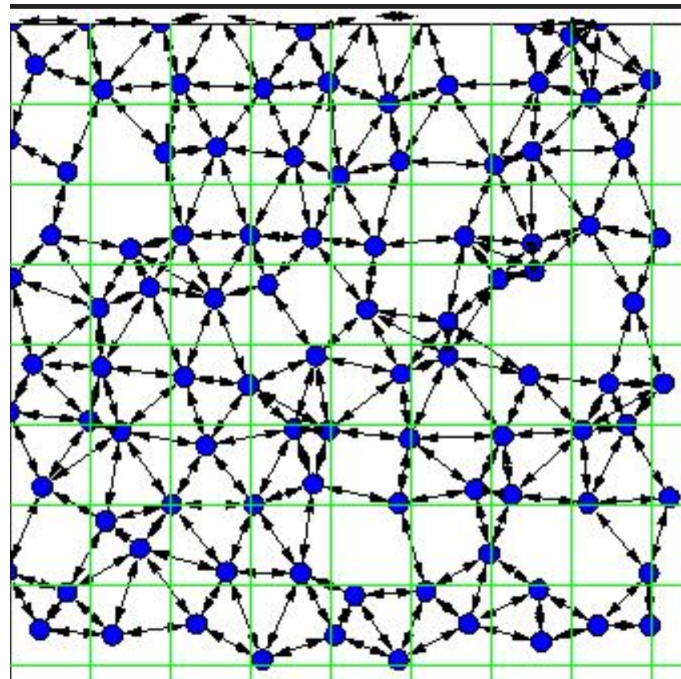


Fig. 7.2. Connectivity of the graph

### a1) Effect of throughput against background traffic

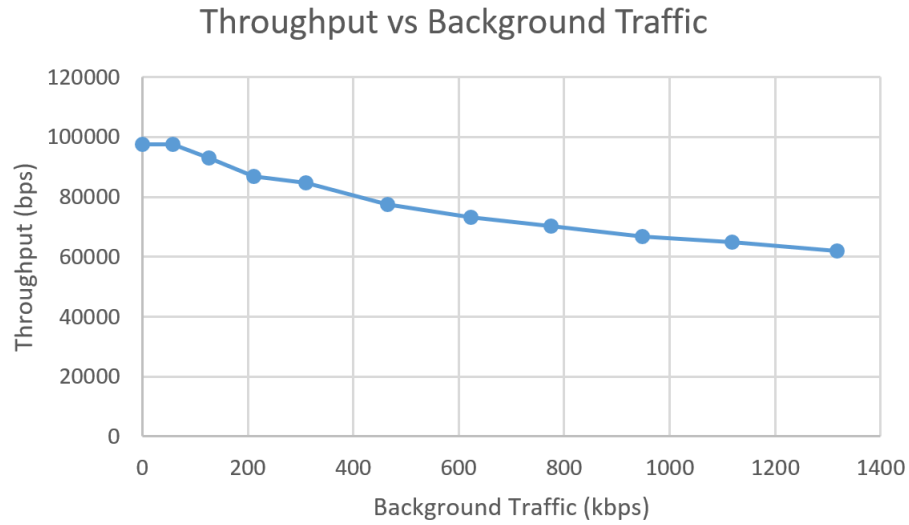


Fig. 7.3. Throughput against background traffic

The traffic flow from genuine source and destination nodes as shown in Figure 7.1 are probed throughout the simulation time. It is observed that as the background traffic overwhelms the network resources, Ant Colony algorithm becomes more sensitive to background traffic. As a result, there's a dip in throughput when background traffic increases.

### a2) Effect of delay against background traffic



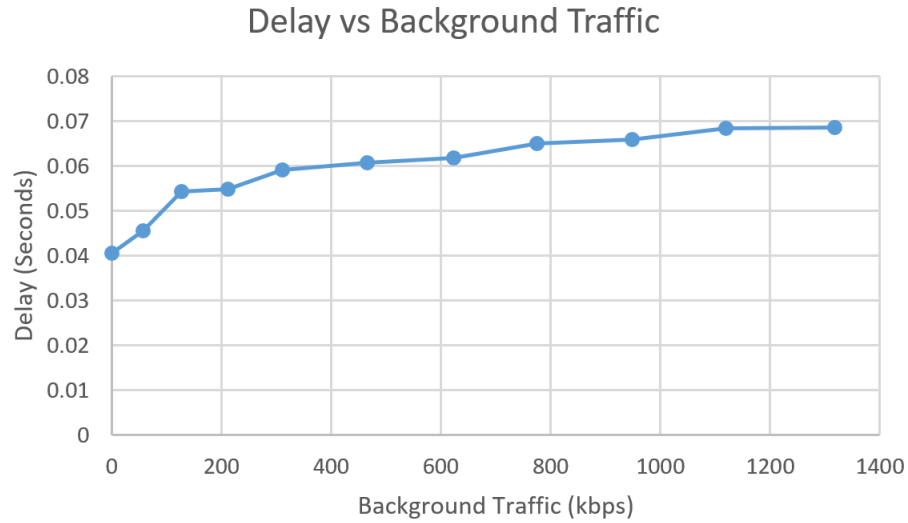


Fig. 7.4. Delay against background traffic

Relationship between delay and background traffic is very much linear. Since ant colony forms multiple paths with varying hop counts between source and destination pair, these paths are affected by background traffic which brings in the latency in the system.

### a3) Effect of Packet loss against background traffic

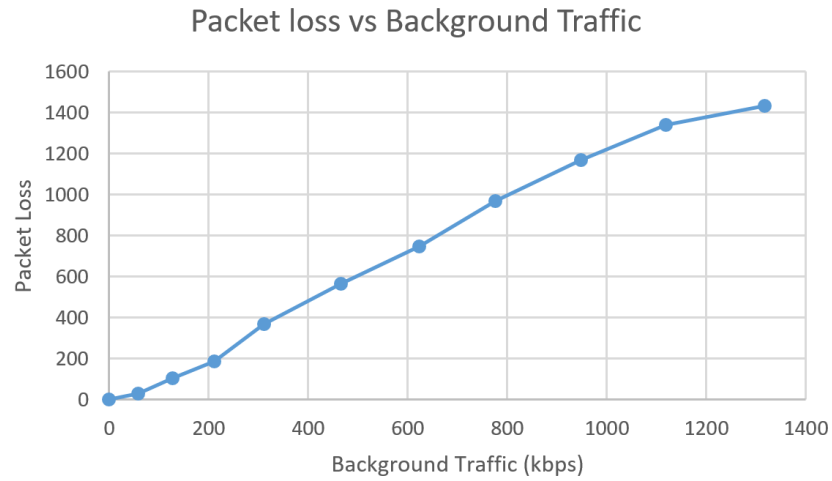
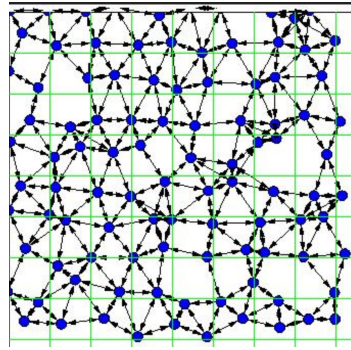


Fig. 7.5. Packet loss against background traffic

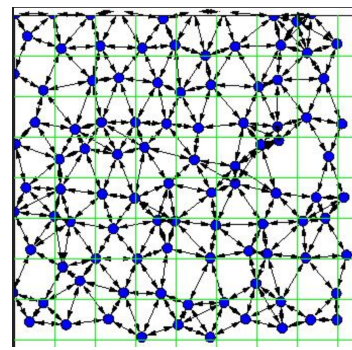
Packet loss exponentially increases with increase in background traffic. Conditions like packet collision, congestion, overhearing due to background load affects multiple paths between source and destination pair which makes Packet losses more sensitive to background traffic.

### b) Average Degree

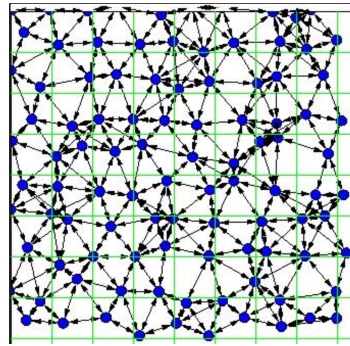
The topology and complexity of network varies on the transmission range of nodes. Before proceeding for the analysis, it is vital for us to understand the topology changes and degree distribution when the transmission range is varied. In this experiment, the transmission range is varied from 71 meters to 120 meters. Figure 7.6 - 7.7 shows the connectivity of graph as well as its complexity when the transmission range is varied in increasing order.



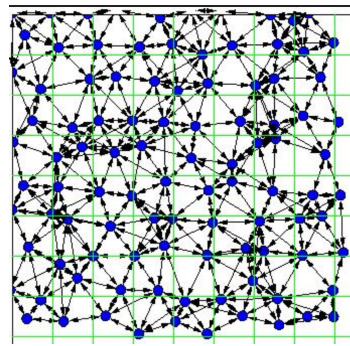
(a) Average Degree 4.94 (Tx  
Range: 71 m)



(b) Average Degree 5.34 (Tx  
Range: 75 m)

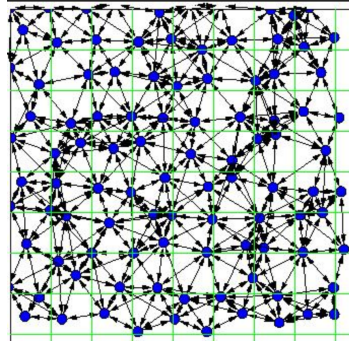


(c) Average Degree 5.94 (Tx  
Range: 81 m)

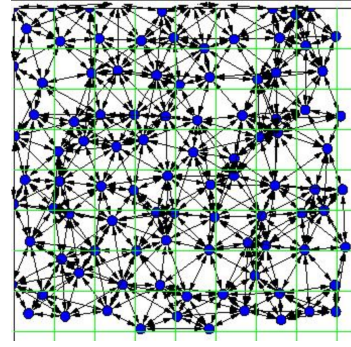


(d) Average Degree 6.94 (Tx  
Range: 87 m)

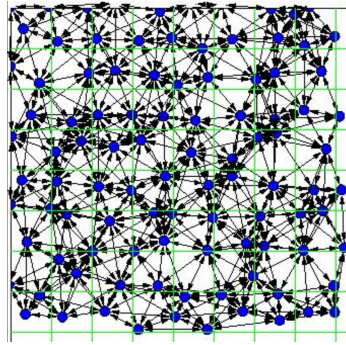
Fig. 7.6. Topology and connectivity of network.



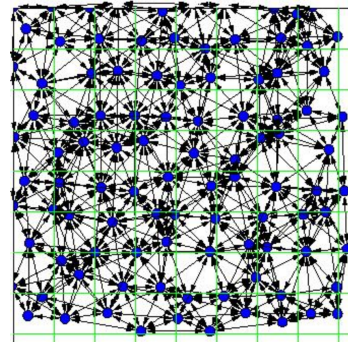
(a) Average Degree 8.03 (Tx  
Range: 93 m)



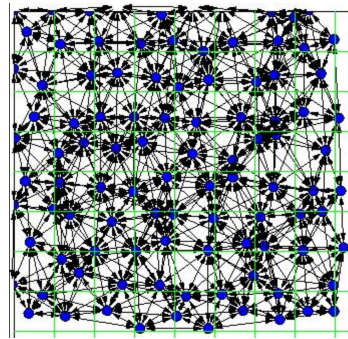
(b) Average Degree 9.42 (Tx  
Range: 99 m)



(c) Average Degree 10.54 (Tx  
Range: 105 m)



(d) Average Degree 11.63 (Tx  
Range: 111 m)



(e) Average Degree 13.43 (Tx  
Range: 120 m)

Fig. 7.7. Topology and connectivity of network continued.

In the following figures the degree distribution will be shown against every average degree of network.

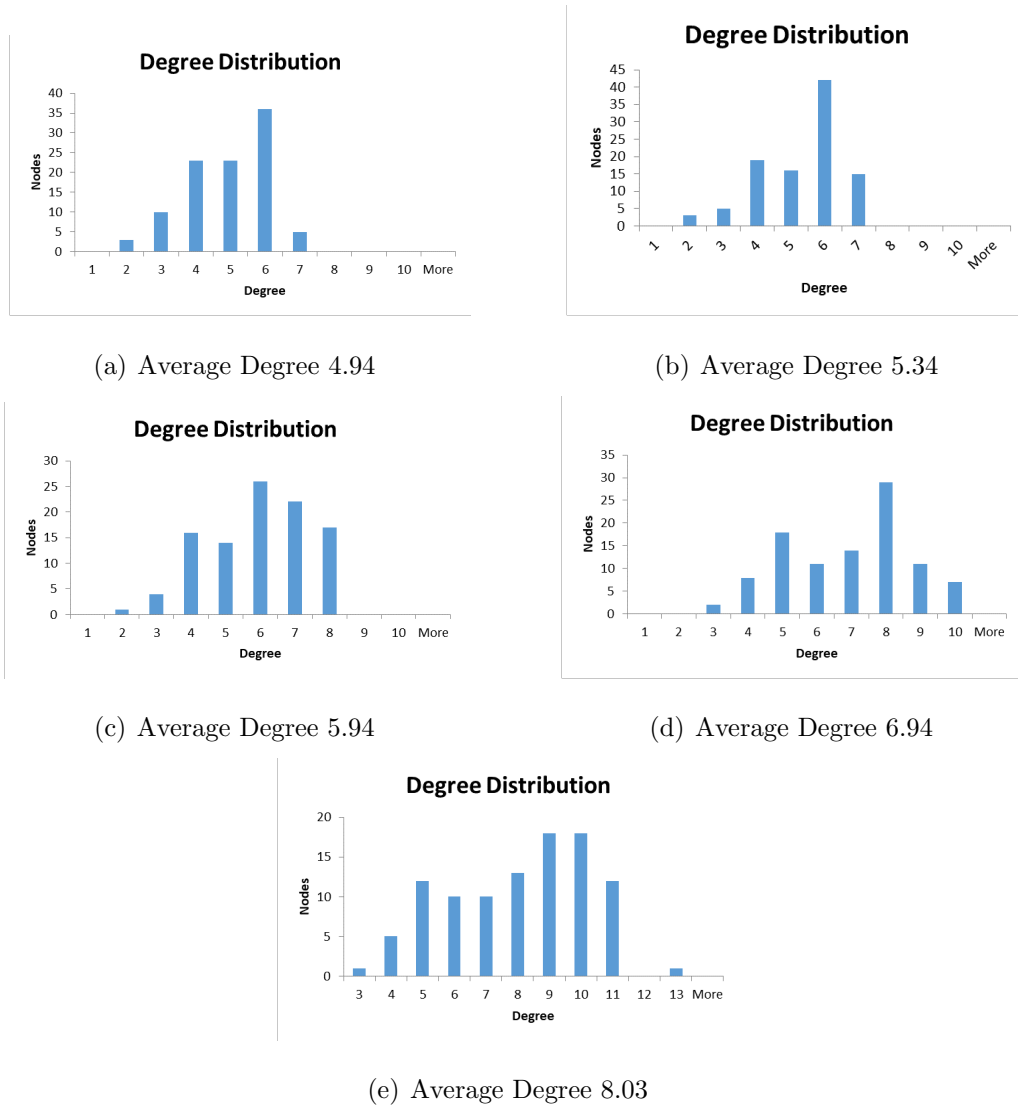
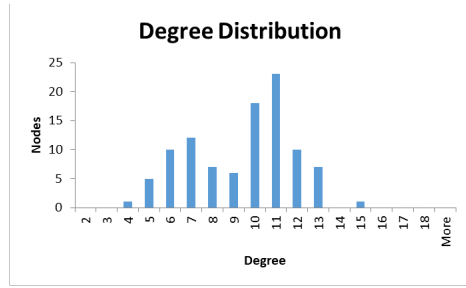
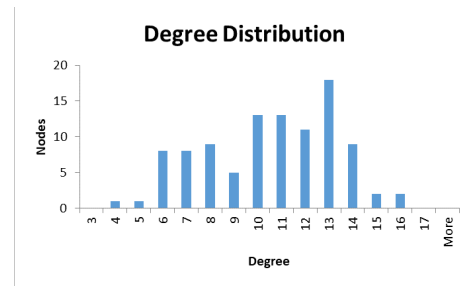


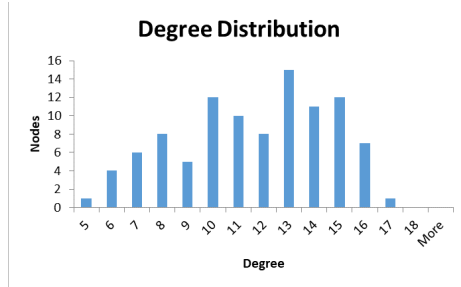
Fig. 7.8. Degree Distribution against varying transmission range



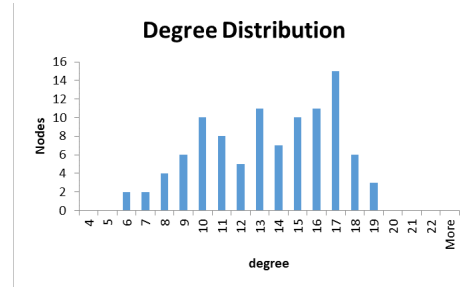
(a) Average Degree 9.42



(b) Average Degree 10.54



(c) Average Degree 11.63



(d) Average Degree 13.43

Fig. 7.9. Degree Distribution against varying transmission range (continued)

### b1) Analysis of Throughput against average degree

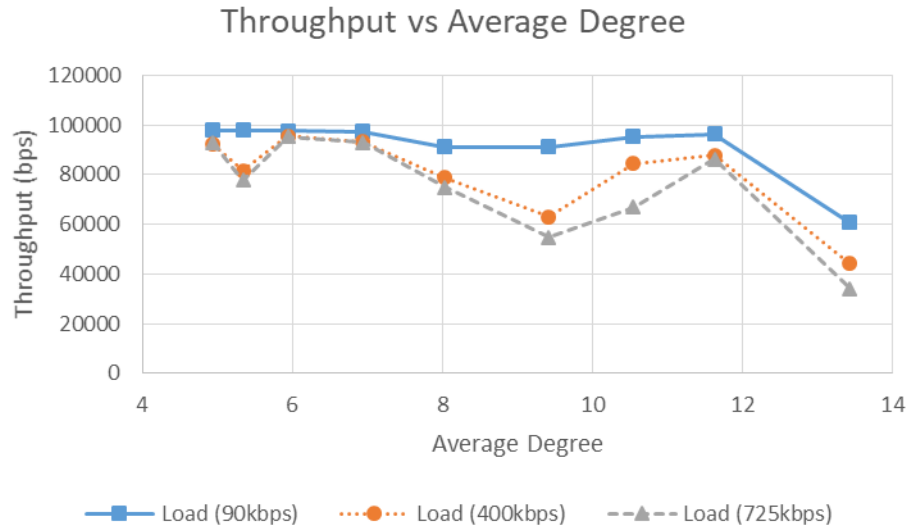


Fig. 7.10. Throughput vs Average Degree

The Throughput decreases with increase in average degree of the network. As the transmission range increases, the complexity of network increases as shown in Figures 7.6- 7.7, which leads to packet collisions and high interference. It has been observed that when the average degree of network is close to 6, the network experiences its best performance as the throughput of network is unaffected by the background load possibly due to best connectivity in the network.

## b2) Analysis of Delay against average degree

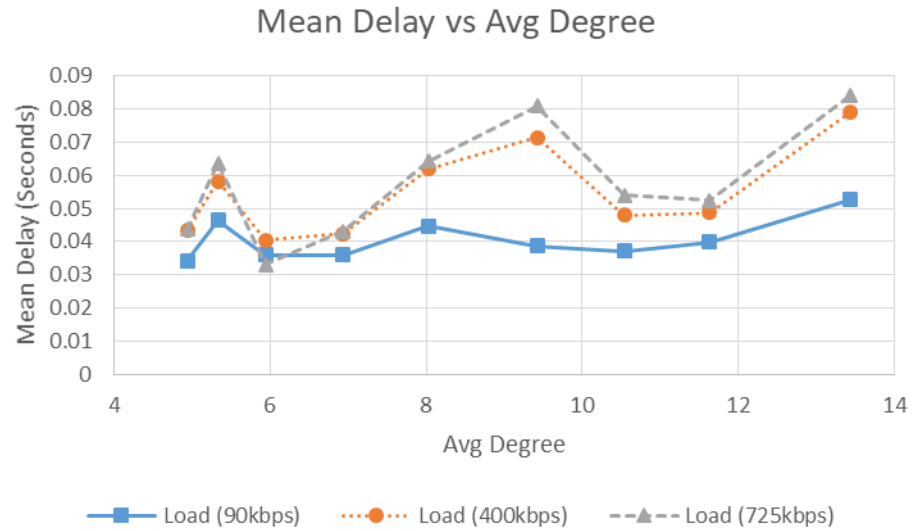


Fig. 7.11. Delay vs Average Degree

It is observed that mean delay is directly proportional to average degree of network. With increase in average degree of node, the node density becomes higher, which means a node can access more number of neighbors around itself. This leads to increase in overhearing and congestion which is the root cause for increase in delay. However the optimal average degree of network is close to 6 as the delay of network is hardly affected.



### b3) Analysis of Packet Loss against average degree

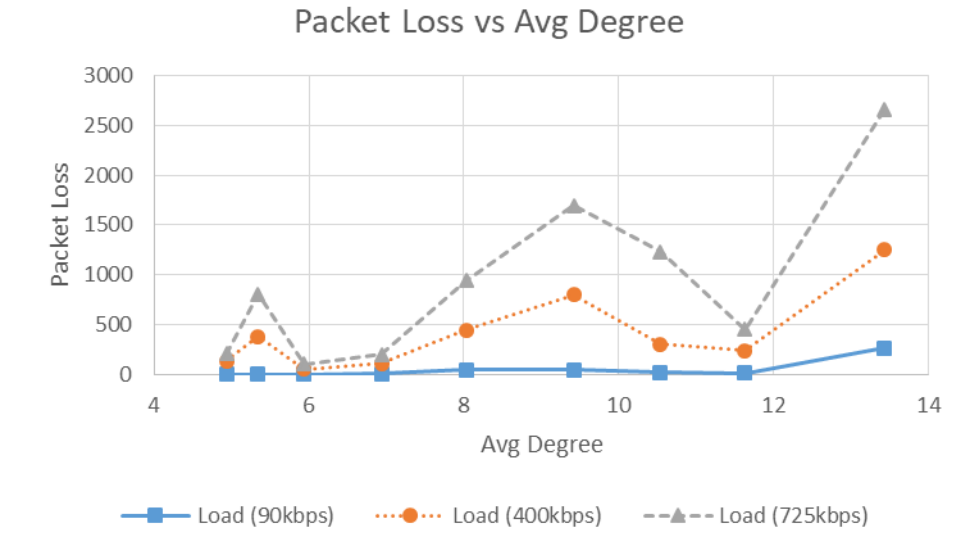


Fig. 7.12. Packet Loss vs Average Degree

Packet loss is expected to increase with increase in node density due to overhearing and congestion. The packet loss is found to be the least when the average degree of network is around 6 as it offers best connectivity.

#### 7.2.2 Randomness Property of Ant Colony Routing Protocol in terms of Hop Count

This section claims about the random hop count property of our routing algorithm by showing the 95% confidence interval. Against every minimum path between source and destination node, the mean hop between them is calculated along with the confidence interval that gives a range of hop count values with 95% surety.

The nodes here are placed on 10x10 2-D grid as shown in Figure 7.13. This uniform topology of network is chosen because it makes sure that majority of the nodes have equal degree around itself.

Otherwise in case of random placement of nodes, it is possible to have a bias in hop count performance as the degree of nodes are not consistent.

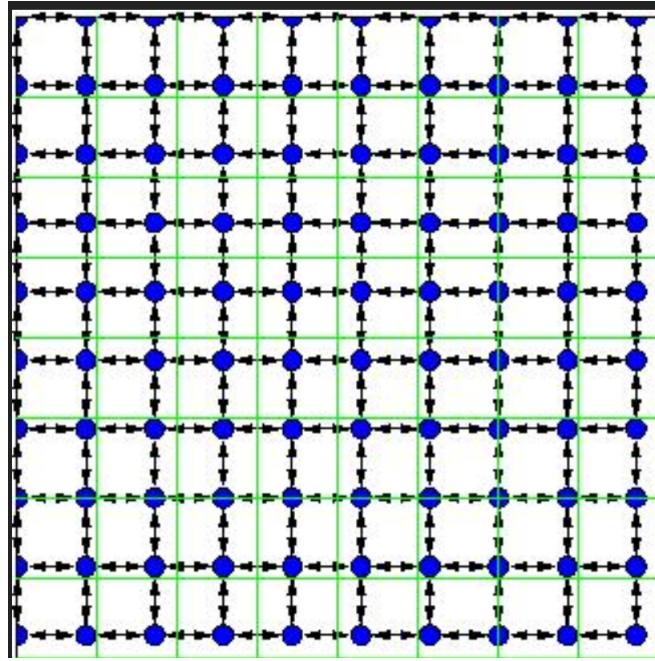


Fig. 7.13. Placement of nodes in a square grid

The Figure 7.14 shows the confidence interval of hop counts between the source and destination node.

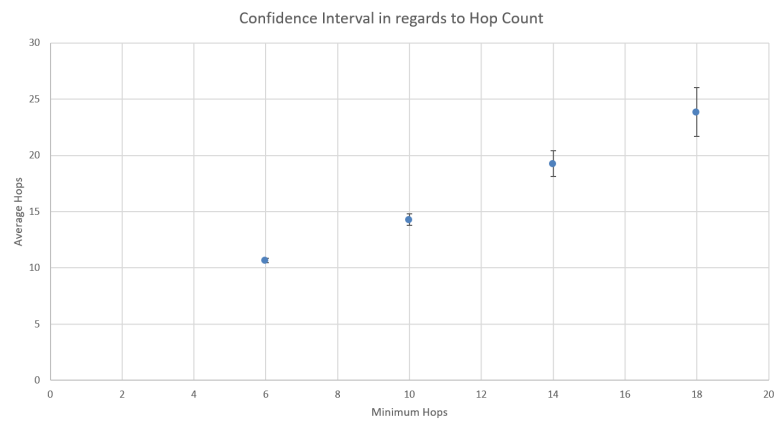
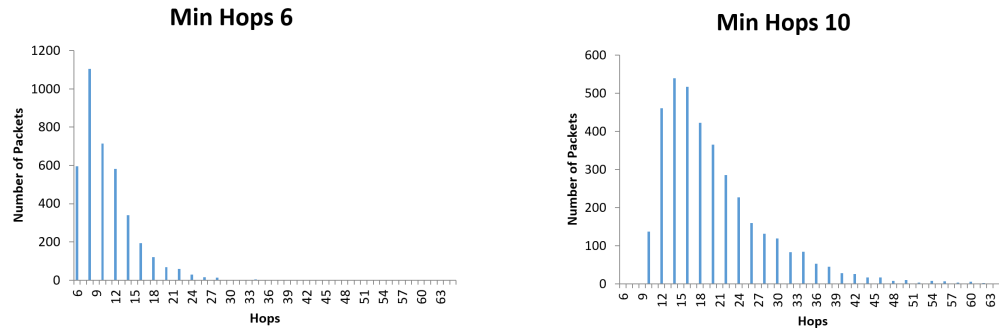


Fig. 7.14. Confidence interval in regards to hop count

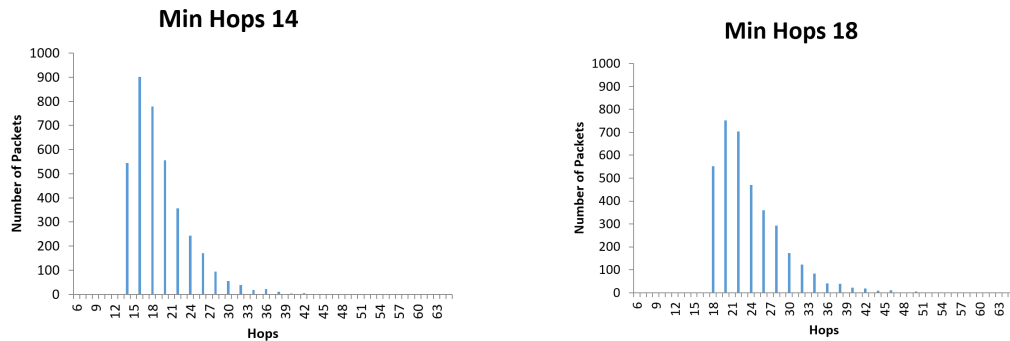
It can be inferred that when the source node and destination node are placed such that the minimum number of hops between them, let's say is 18, there's 95% chance that the packets received by the destination node will have hop count in the range from 22 to 27 hops.

The Figure 7.15 shows the hop count distribution for every minimum hops shown in Figure 7.14.



(a) Hop Count Distribution for min hops 6

(b) Hop Count Distribution for min hops 10



(c) Hop Count Distribution for min hops 14

(d) Hop Count Distribution for min hops 18

Fig. 7.15. Hop Count Distribution against minimum hops

### 7.3 Importance of impact factors in Ant Colony Algorithm

This section discusses about the need of individual impact factors taken into the consideration as described in section 6.

#### 7.3.1 RSSI as an impact factor

Using RSSI is a well known technique to measure distance between source and receiver. The distance is estimated by using the strength of received wireless signal. The relationship between RSSI and distance is inversely proportional as shown in Appendix A. Low values of RSSI cause more number of packet losses which in turn makes failure of transmissions of packets. The motive behind this analysis is to find the total number of failed transmission of data packet by varying the weightage of RSSI coefficient. Each node gathers this data from MAC (L-2) layer.

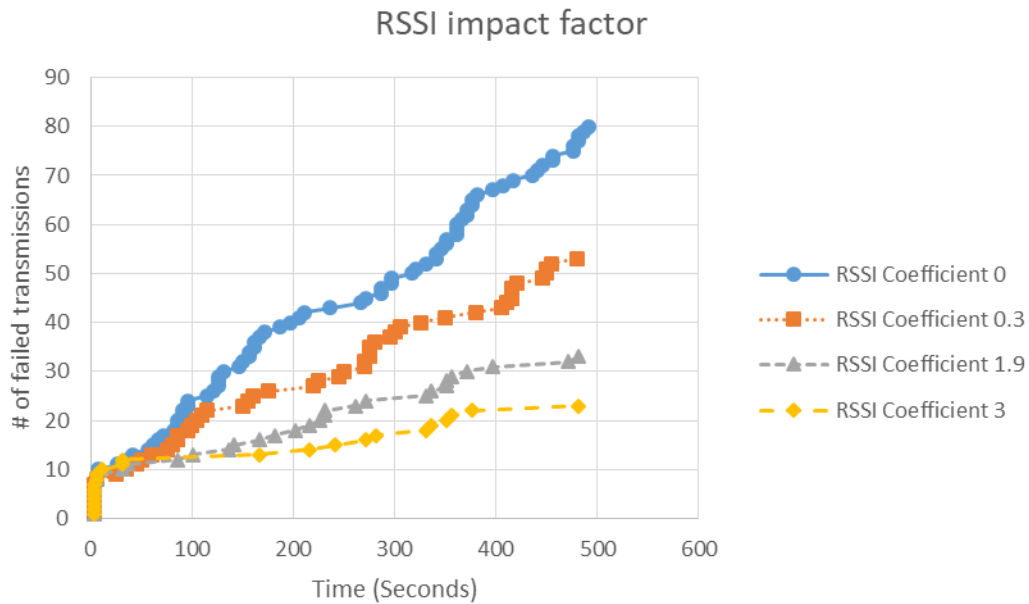


Fig. 7.16. Importance of RSSI Coefficient factor

The figure above shows how the failed transmission of packets can be controlled with RSSI impact factor. RSSI Coefficient 0 signifies there's no consideration of RSSI in the routing algorithm due to which maximum failure of packet transmissions are observed. When the RSSI is taken into consideration by increasing the weightage of RSSI coefficient, the number of failed transmissions of data packets drops down.

### 7.3.2 Hop Count as an impact factor

Hop Count information is used by the nodes to learn about the how far they are from source and destination. This knowledge gathered from this data helps to reduce the end-to-end delay in routing as much as possible.

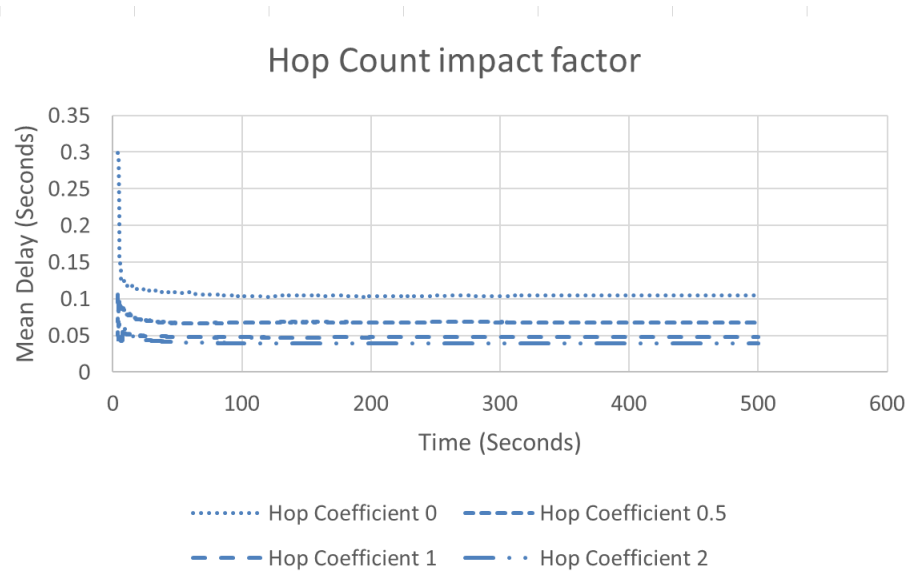


Fig. 7.17. Importance of Hop Count factor

With reference to the figure given above, highest delay is observed when hop count factor is turned off (hop coefficient 0) as the routing protocol has no knowledge about the source and destination.

When the preference to hop count factor is increased by changing the hop count coefficient, the nodes are aware of source and destination in terms of hop counts and the reduction in end-to-end delay is expected. Overall 60% reduction in delay is observed.

### 7.3.3 Residual Energy as an impact factor

The inclusion of residual energy as an impact factor in Layer 3 (IP layer) routing makes the protocol energy aware as the routing decision considers the residual energy among neighboring nodes while forwarding the data packet. If this impact factor is excluded from routing, the reduction in network lifetime is expected.

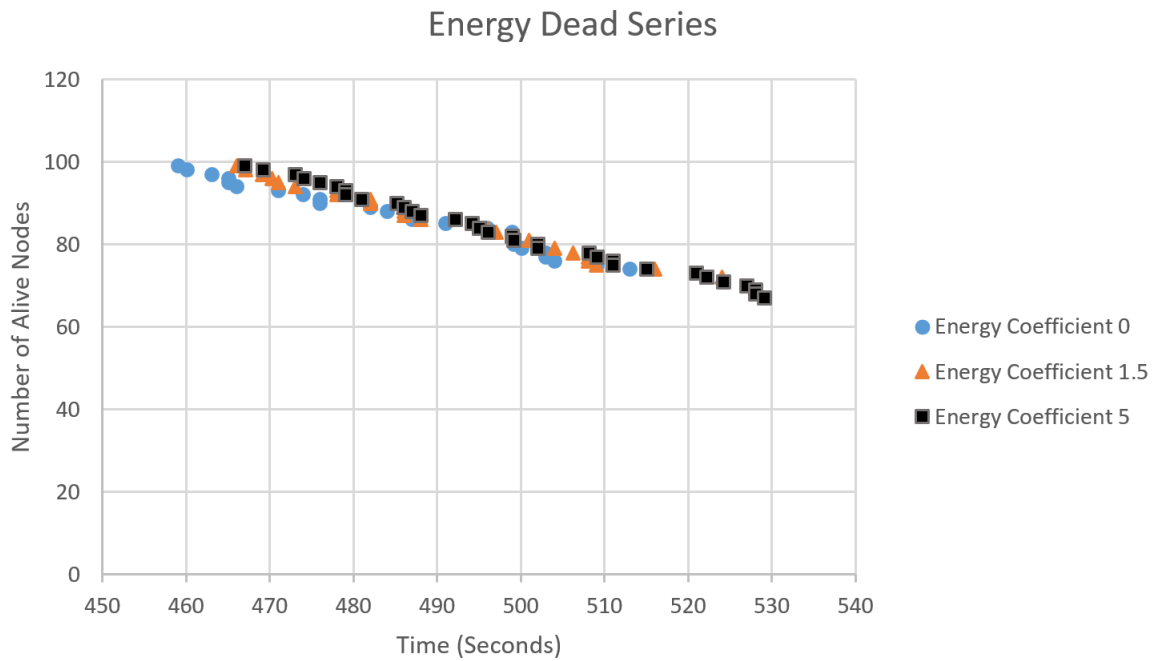


Fig. 7.18. Importance of Energy factor

In the figure given above, the time at which the node dies is recorded for every node which gives us the relationship between number of alive node versus time. The source transmits data at every second. When the energy impact factor is not considered (Energy Coefficient 0), the first node dies at 459 seconds and the communication link

breakage occurs at around 515 seconds, also known as network partition time where source and destination are no longer able to communicate with each other.

When slight weightage is given to Energy impact factor (Energy coefficient 1.5), network partition occurs at 524.14 seconds and when energy coefficient is 5, highest energy related performance is observed where the network partition time is around 530 seconds.

Overall 15 seconds of improvement in network partition time is observed when energy impact factor is taken into consideration.

## 8. COMPARATIVE ANALYSIS

This section compares Ant Colony Routing Algorithm against the AODV Protocol. Various test scenarios such as background traffic analysis, varying transmission range, energy analysis, etc. are considered for comparative purposes.

### 8.1 Background Traffic

#### 8.1.1 Throughput effect against background traffic

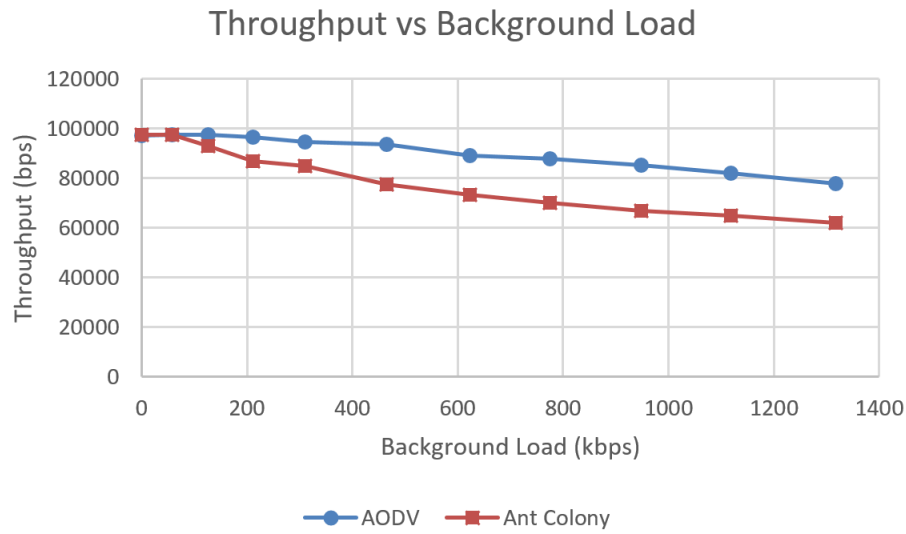


Fig. 8.1. Throughput Comparison

It is deduced that Ant Colony is more sensible to background traffic as compared to AODV as it is evident from the throughput trend. Since AODV forms path between source and destination pair with least number of hops unlike Ant Colony, it suffers less packet loss comparatively which makes the throughput performance better than Ant Colony.



### 8.1.2 Effect of Mean Delay against background traffic

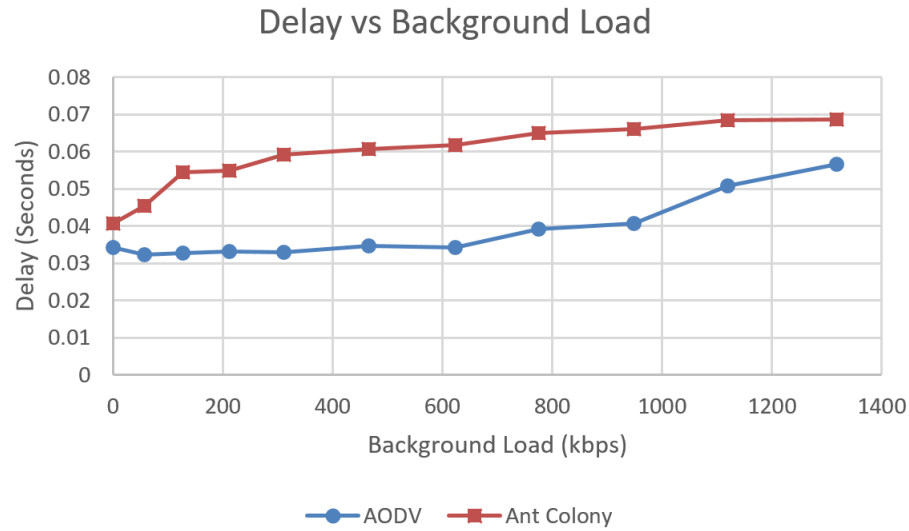


Fig. 8.2. Mean Delay Comparison

AODV experiences lesser delay as compared to Ant Colony protocol. The reason is that AODV forms the shortest path between source and destination. As a result, even though delay in AODV increases with background traffic, its going to be lesser than that of Ant Colony. Ant Colony Algorithm is not designed for routing over the best path between source and destination which makes the performance of AODV better than Ant Colony in terms of delay.

### 8.1.3 Effect of Packet Loss against background traffic

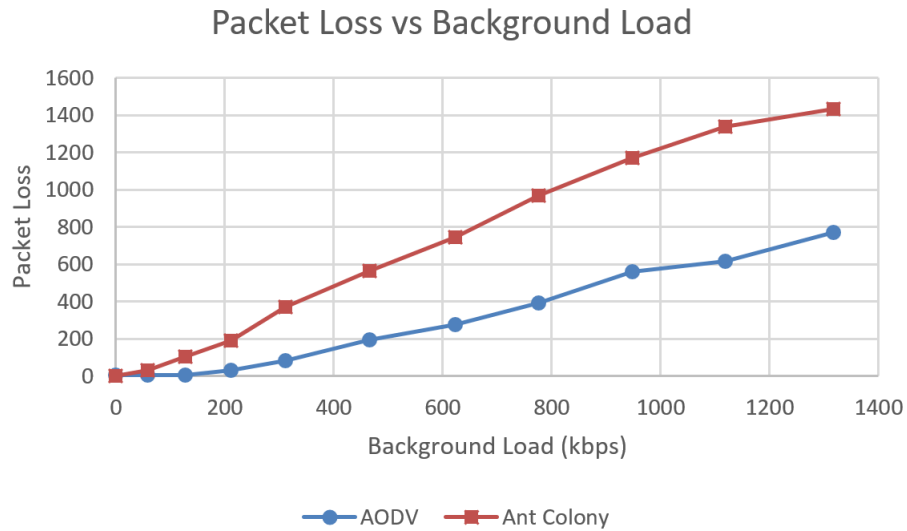


Fig. 8.3. Packet Loss Comparison

Ant Colony suffers more packet loss against increase in background traffic when compared to AODV. Along with optimal hop path between source and destination, Ant Colony constructs multiple paths with higher hop counts. Therefore the probability of packet losses along higher hop count paths is more than AODV which only constructs hop optimal path. This makes Ant Colony more prone to packet losses.

## 8.2 Average Degree

As explained in previous Section 7.2.1, the topology is made dense by changing the transmission range of all the nodes and the performance metrics are measured.

### 8.2.1 Throughput effect against average degree

By including the background load in the system, the throughput is measured against varying transmission range and then the following observations are made.

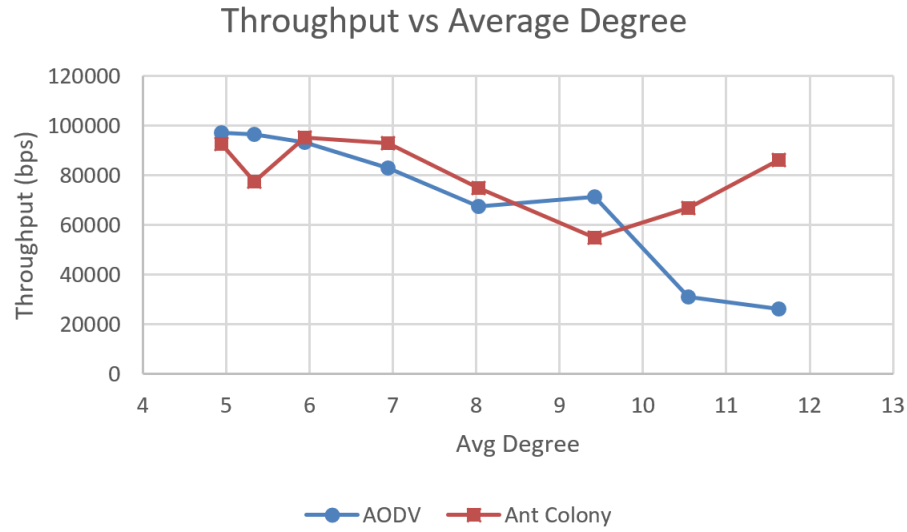


Fig. 8.4. Throughput Comparison against average degree

Ant Colony attains maximum throughput at average degree of 6 whereas AODV has its maximum throughput at average degree of 4.94. During increase in transmission range, AODV experiences more packet loss as shown in figure 8.5 due to network congestion which makes the throughput performance lower than that of Ant Colony.

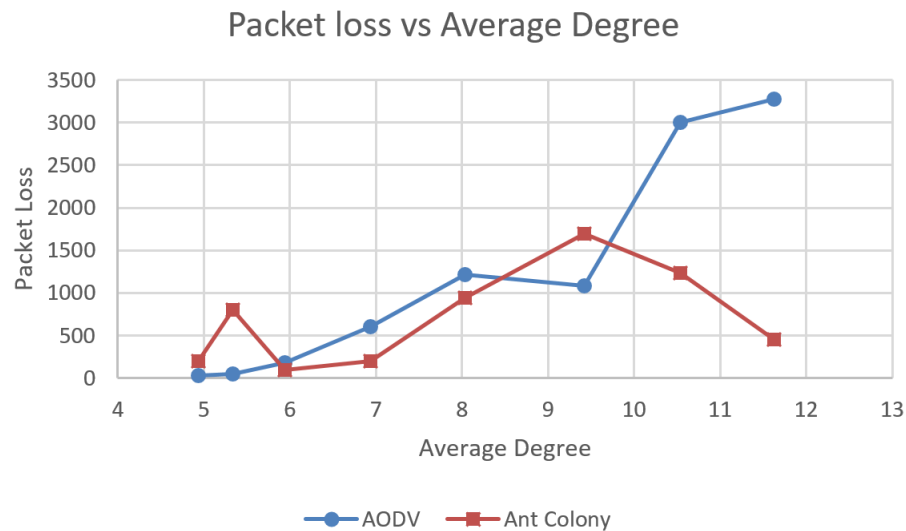


Fig. 8.5. Packet loss against average degree

### 8.2.2 Effect of Mean Delay against Average Degree

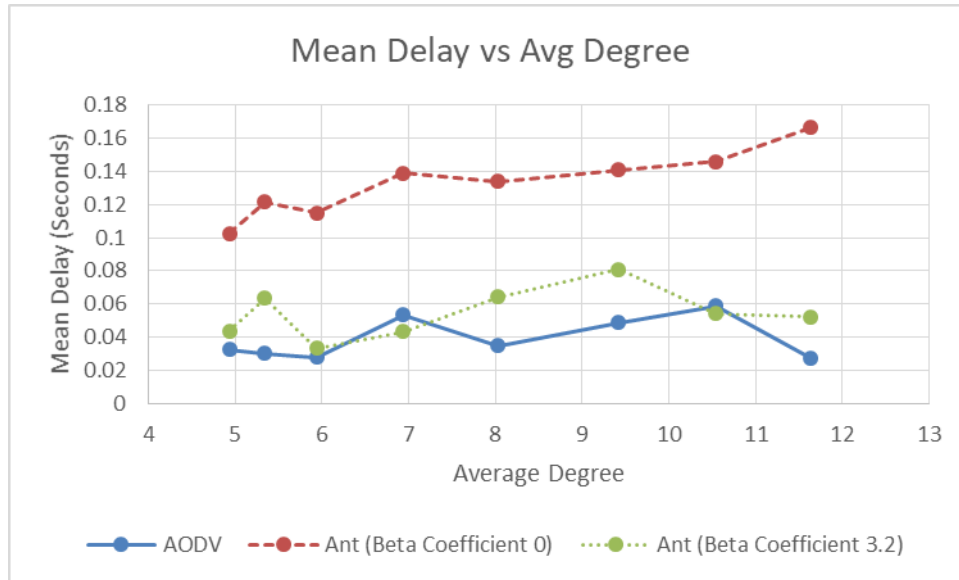


Fig. 8.6. Mean Delay comparison against average degree

AODV outperforms Ant Colony when it comes to delay against increase in transmission range. As AODV forms shortest path between source and destination, with increase in transmission range, packets from source node can reach destination with fewer number of hops making the delay comparatively less. Also in case of Ant Colony, maximum delay is observed that when the hop count impact factor is ignored as the nodes ignore the hop count information in its routing table.

### 8.3 Energy Comparison

In both AODV and Ant Colony, all nodes are equipped with battery. With reference to data-sheet of FRDM KW41Z [19] micro-controller, the transmitting current has been configured to 6.1 mA, the receiving current is set to 6.8 mA. The micro-controller uses a single coin cell battery [20] which has a idle current capacity of 0.19 mA. The goal in this section is to study the energy analysis and lifetime of the network.

### 8.3.1 Energy Depletion Series

Communication between source and destination takes place at an interval of 2 minutes in order to mimic the wireless sensor network applications. Figure 8.7 shows the order at which the nodes become energy depleted when two different routing protocols are used.

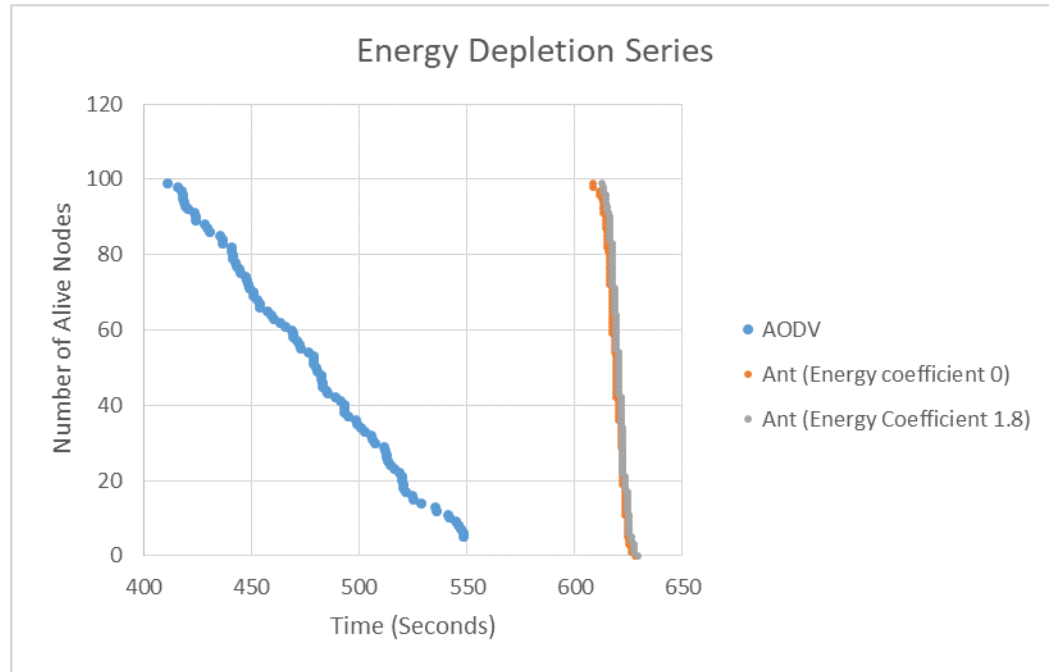


Fig. 8.7. Comparison of dead node series

The above figure shows that while using AODV routing protocol the time at which the first dead node is observed at approximately 411 seconds whereas in Ant Colony, the time at which the first dead node (when  $\delta$  is 1.8) is observed at approximately 612 seconds. When  $\delta$  is 0, time at which the first dead node is observed at 608 seconds. Partition time when AODV is used is observed at 549 seconds whereas when using Ant Colony, partition time is 629 seconds. Thus Ant Colony experiences 49% improvement compared to AODV. The inclusion of residual energy as an impact factor in ant colony routing algorithm makes it more better than AODV in terms of energy performance.

### 8.3.2 Comparison of Remaining Energy over different Time-stamps

The following figures have been drawn using MATLAB [21] color plots where RGB value is varied according to the residual energy of node at that time instant. The colorbar indicates darker color as full energy and as the node loses energy, the color gradually changes to brighter shade.



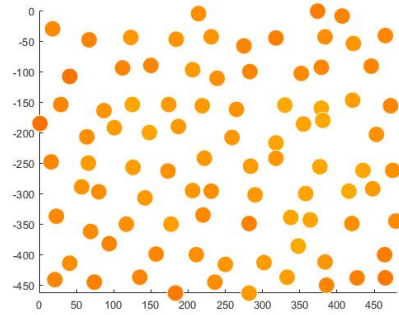
Fig. 8.8. Color bar of remaining energy

This Section visualizes the remaining energy of nodes over different time intervals which gives the reader a rough idea about energy consumption when using different routing protocols.

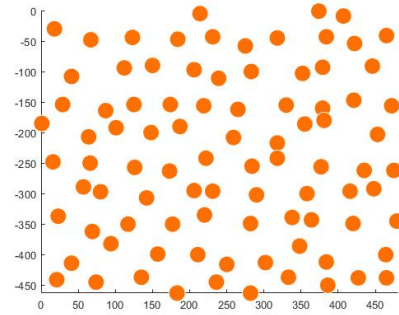


Fig. 8.9. Remaining Energy over different time stamps

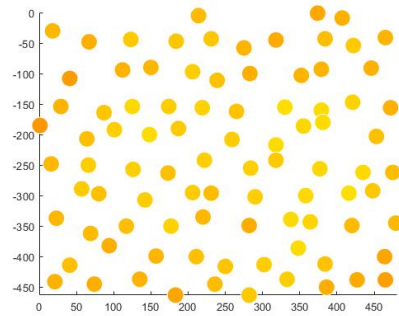
Till 100 seconds, it is hard to observe changes in the residual energy of nodes. When around 200 seconds, nodes begin to fall under middle region of colorbar shown in Figure 8.8 which means that energy level of nodes is going down.



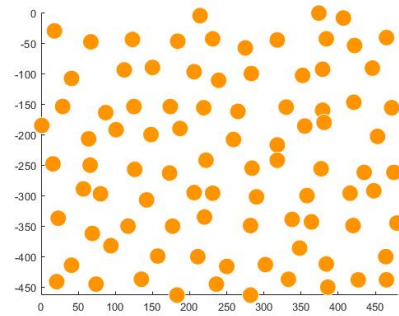
(a) AODV at 300 seconds



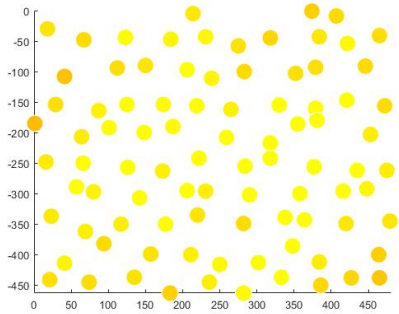
(b) Ant Colony at 300 seconds



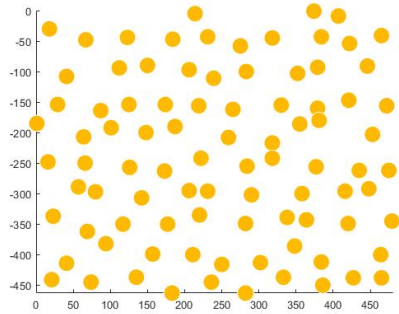
(c) AODV at 400 seconds



(d) Ant Colony at 400 seconds



(e) AODV at 500 seconds



(f) Ant Colony at 500 seconds

Fig. 8.10. Remaining Energy over different time stamps (continued)

At 400 seconds, it is observed that in case of AODV protocol, the nodes around the middle of topology have average remaining energy of 0.0699 Joules with standard deviation of 0.0127 Joules as compared to the rest of nodes which has average remaining energy of 0.1265 Joules with standard deviation of 0.0173 Joules. Whereas in ant



colony, the nodes around middle region have average remaining energy of 0.166 Joules with standard deviation of 0.0008 Joules as compared to the rest of nodes having average residual energy of 0.1683 Joules with standard deviation of 0.001 Joules. At around 500 seconds, most of the nodes using AODV have been energy depleted except few of them on the extreme sides. In case of ant colony, the nodes can communicate for longer time due to relatively high residual energy.

Two things can be claimed with reference to the Figures 8.9 and 8.10. While using Ant Colony protocol, the nodes at any time instant and irrespective of their location (extreme corners, middle of topology) have evenness in regards to residual energy. This is due to the fact that Ant Colony uses remaining energy as one of the impact factor during packet forwarding process. Also the ability to establish multiple paths between source and destination makes it consume less energy unlike AODV which follows hop optimal approach leading to discrepancy in terms of residual energy of nodes.

Figures below show the remaining energy of nodes at 600 and 640 seconds when using Ant Colony routing protocol. AODV could not run for longer period of time because of high energy consumption.

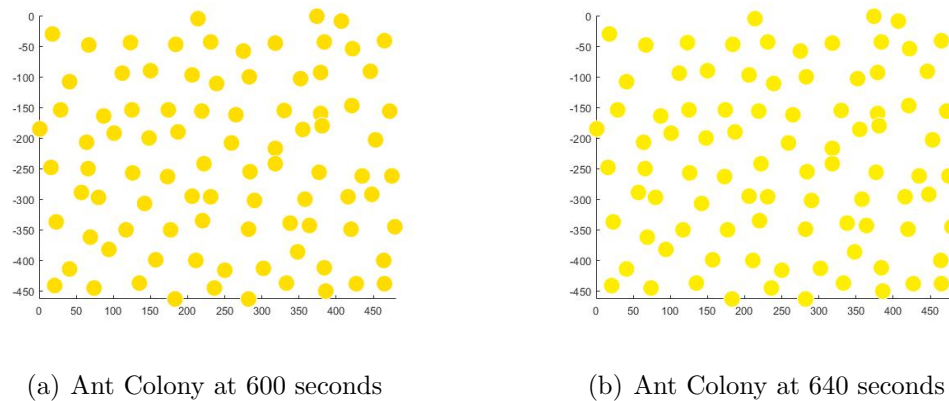


Fig. 8.11. Ant Colony time stamps at 600 and 640 seconds

When time is at 640 seconds, all the nodes are uniformly used and are energy depleted as can be seen in the figure shown above.

### 8.3.3 Comparison of standard deviation of residual energy

Starting from time 50 seconds, at regular intervals of 25 seconds, the standard deviation of residual energy for all 100 nodes in the network is calculated and is plotted against time as shown in Figure 8.12 below.

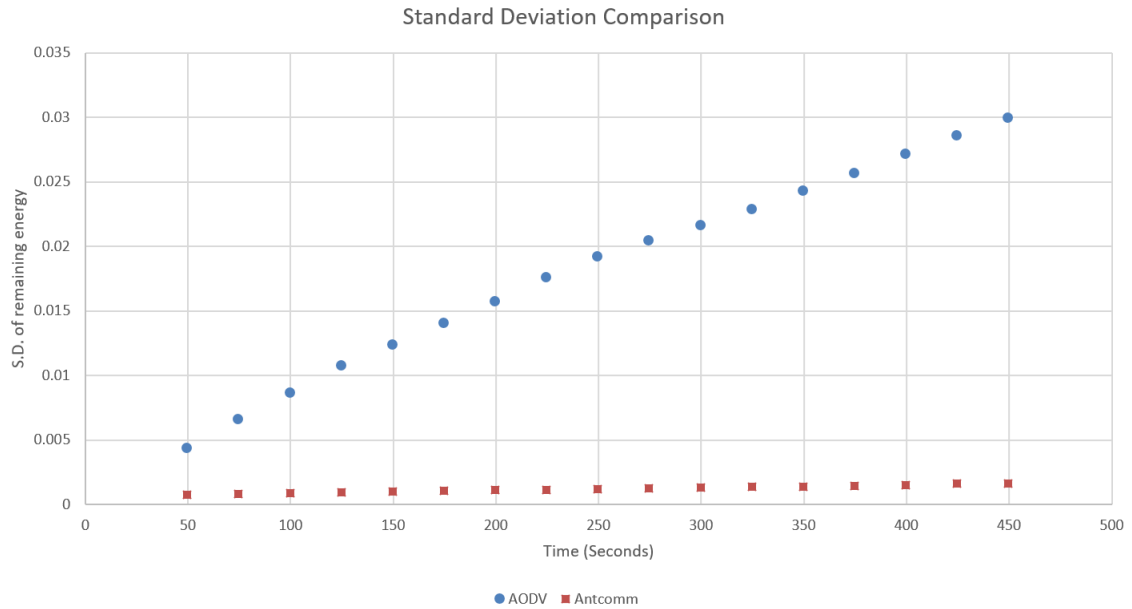


Fig. 8.12. Comparison of Standard Deviation of Residual Energy

It is observed that standard deviation of AODV is very high compared to standard deviation of Ant Colony Routing. The reason is in AODV routing, there's only one path between source and destination node. As a result, the nodes along the chosen path are used more compared to the rest of the nodes in the network. This creates a disparity among nodes in terms of residual energy.

Whereas in Ant Colony Routing, there are multiple paths between source and destination. Due to this, most of the nodes are used uniformly during the communication process which makes the standard deviation low due to evenness in remaining energy.

### 8.3.4 Comparison of Average Remaining Energy

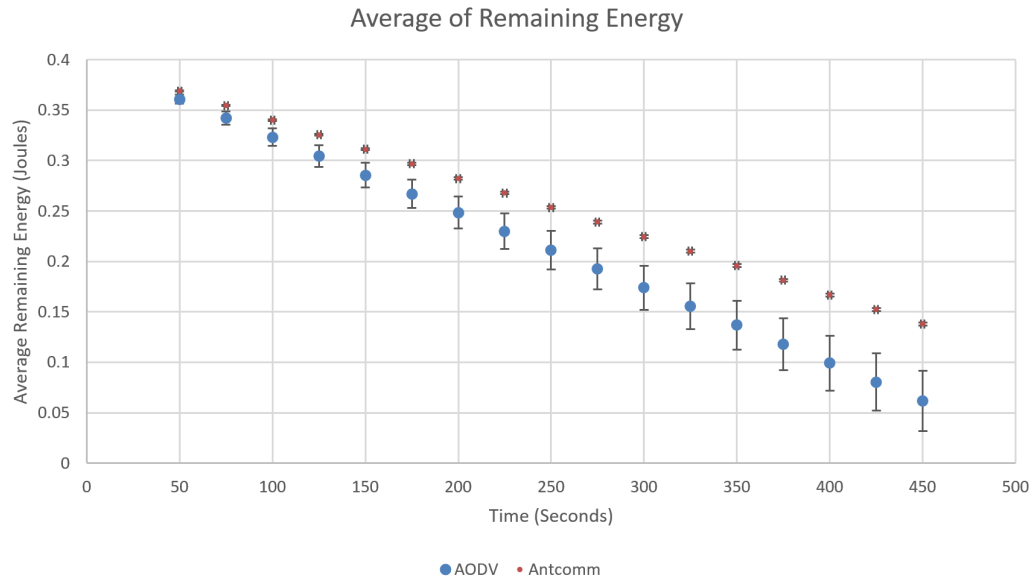


Fig. 8.13. Comparison of average remaining energy

The difference in average remaining energy of ant colony algorithm when compared against AODV is quite high. AODV, being a single path routing algorithm, consumes more energy over a period of time unlike Ant Colony algorithm which constructs multiple paths and has a packet forwarding mechanism which selects the next hop relative to the proportion of impact factors as explained in Section 6. The error bars around the average energy data point shows the standard deviation of residual energy.

## 8.4 Hop Count Comparison

Hop Count analysis is studied in this section each time by changing the transmission range of nodes. As shown in the figure below, for every time interval, the average of hop counts for first 30 data packets received by destination are calculated and is compared.

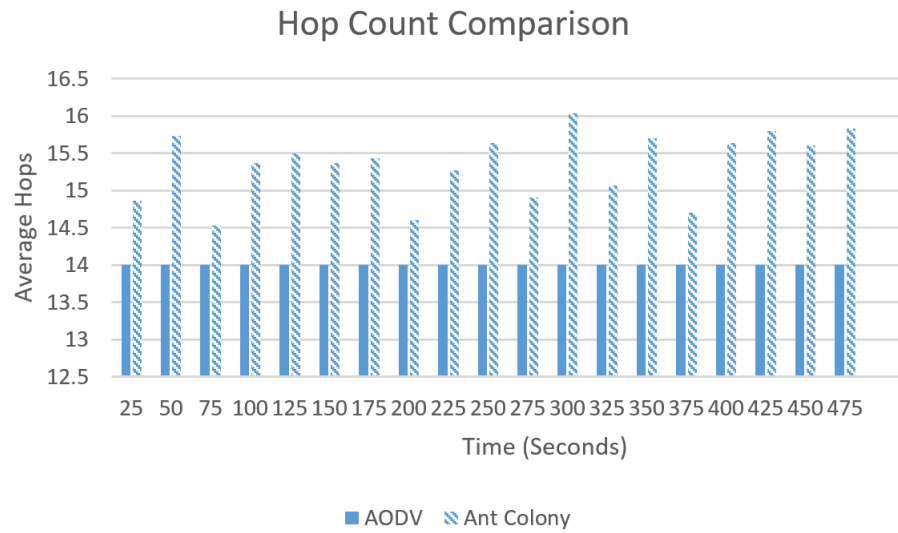


Fig. 8.14. Hop Comparison when Avg degree is 4.94

It can be claimed from these figures that while using AODV routing protocol, once source establishes route towards destination, it uses the same path no matter how long the communication takes place because it follows hop optimal approach.

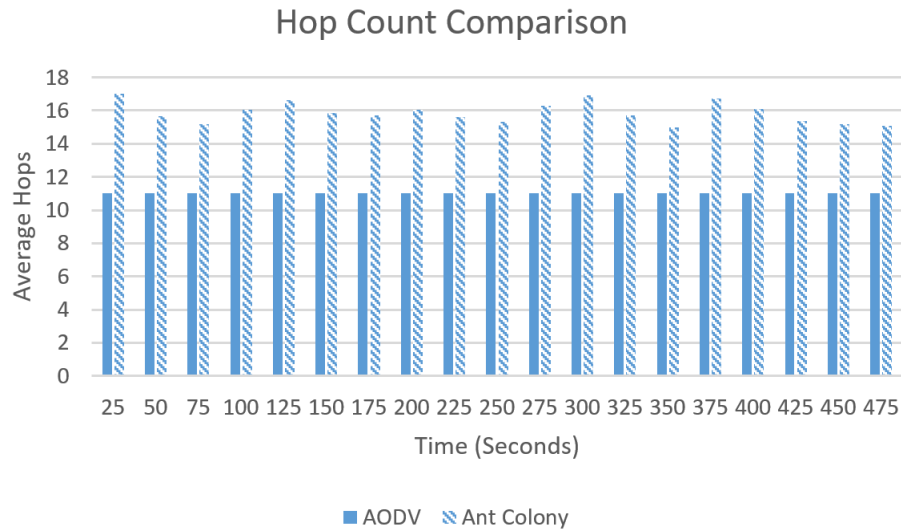


Fig. 8.15. Hop Comparison when Avg degree is 5.94

With increase in average degree of network it is observed that when AODV protocol is used, there is reduction in average hops of packets from source to destination as AODV uses single shortest path route which makes it hop optimal. Whereas Ant Colony protocol selects the next hop relative to the proportion of impact factors which involves randomness. Hence ant colony protocol is not hop optimal.

## 9. CONCLUSION AND FUTURE WORK

Comprehensive performance analysis of Ant Colony routing algorithm under various load conditions, density of network, etc is carried out in this research. First the challenges faced by sensor networks are described in Section 1.1. The methods to overcome these challenges are elaborated in Section 6 and in Section 7.3 the performance of network is shown by varying the weightage of individual impact factors. Hop distribution and its confidence interval gives the idea of how data packets are forwarded across multiple paths between source and destination.

Performance of Ant Colony routing algorithm was analyzed and compared against AODV routing by considering various performance metrics such as throughput, delay, packet loss, residual energy, etc. In terms of energy analysis, Ant Colony outperforms AODV as the goal for Ant Colony routing is to extend the lifetime of network by constructing multiple paths. There are cases where AODV performs better than Ant Colony routing algorithm since latter involves the decision to forward the packet randomly which doesn't always favour the optimal path unlike AODV, which forms shortest paths between source and destination.

It should be noted that the current version of NS-3 simulator doesn't have the module for Ant Colony routing algorithm. Development of this module required good understanding of C/C++ Programming as well as solid networking background.

In future, error handling mechanism can be introduced in IEEE 802.11 MAC layer by including repellent pheromone as one of the impact factors. Ants detour from their established paths and give emergency signals to their peers when they encounter emergency situations such as an obstacle (stone) placed in between the paths, water flowing through the paths, etc. Repellent pheromone can act like a signal to the nodes in which the failed transmission of data packets can be minimized by preventing the forwarding of packet to bad neighboring node.

## REFERENCES

## REFERENCES

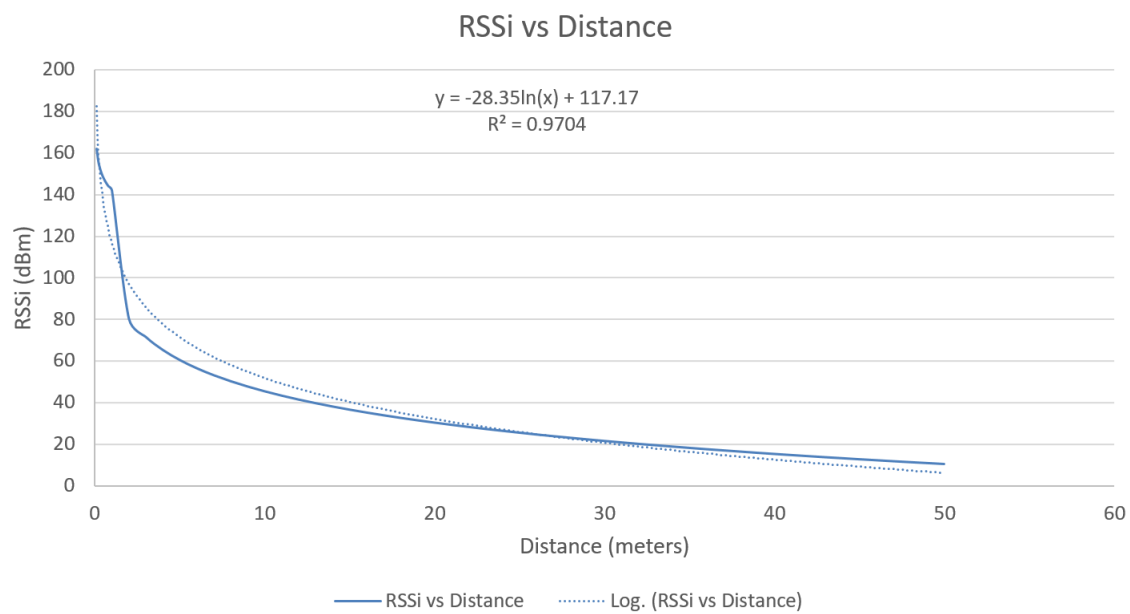
- [1] E. Bonabeau, D. d. R. D. F. Marco, M. Dorigo, G. Theraulaz *et al.*, *Swarm intelligence: from natural to artificial systems*. Oxford university press, 1999, no. 1.
- [2] M. Dorigo and T. Stützle, “Ant colony optimization. 2004,” *Massachusetts Institute of Technology*, 2004.
- [3] G. Di Caro and M. Dorigo, “Antnet: Distributed stigmergetic control for communications networks,” *Journal of Artificial Intelligence Research*, vol. 9, pp. 317–365, 1998.
- [4] T. Camilo, C. Carreto, J. S. Silva, and F. Boavida, “An energy-efficient ant-based routing algorithm for wireless sensor networks,” in *International workshop on ant colony optimization and swarm intelligence*. Springer, 2006, pp. 49–59.
- [5] J.-H. Ho, H.-C. Shih, B.-Y. Liao, and S.-C. Chu, “A ladder diffusion algorithm using ant colony optimization for wireless sensor networks,” *Information Sciences*, vol. 192, pp. 204–212, 2012.
- [6] M. Frey, F. Große, and M. Günes, “Energy-aware ant routing in wireless multi-hop networks,” in *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014, pp. 190–196.
- [7] A. M. A. Elmoniem, H. M. Ibrahim, M. H. Mohamed, and A.-R. Hedar, “Ant colony and load balancing optimizations for aodv routing protocol,” *International Journal of Sensor Networks and Data Communications*, vol. 1, pp. 1–14, 2012.
- [8] K. M. Sim and W. H. Sun, “Ant colony optimization for routing and load-balancing: survey and new directions,” *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 33, no. 5, pp. 560–572, 2003.
- [9] M. Günes, M. Kähler, and I. Bouazizi, “Ant-routing-algorithm (ara) for mobile multi-hop ad-hoc networks-new features and results,” in *Proceedings of the 2nd Mediterranean Workshop on Ad-Hoc Networks (Med-Hoc-Net03)*, 2003, pp. 9–20.
- [10] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc on-demand distance vector (aodv) routing,” Tech. Rep., 2003.
- [11] Z. Zinonos, V. Vassiliou, and T. Christofides, “Radio propagation in industrial wireless sensor network environments: From testbed to simulation evaluation,” in *Proceedings of the 7th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*. ACM, 2012, pp. 125–132.



- [12] J. Kuruwila, A. Nayak, and I. Stojmenovic, "Hop count optimal position-based packet routing algorithms for ad hoc wireless networks with a realistic physical layer," *IEEE Journal on selected areas in communications*, vol. 23, no. 6, pp. 1267–1275, 2005.
- [13] A. Sharma and D. S. Kim, "Robust bio-inspired routing protocol in manets using ant approach," in *International Conference on Ubiquitous Information Management and Communication*. Springer, 2019, pp. 97–110.
- [14] Y. Chapre, P. Mohapatra, S. Jha, and A. Seneviratne, "Received signal strength indicator and its analysis in a typical wlan system (short paper)," in *38th Annual IEEE Conference on Local Computer Networks*. IEEE, 2013, pp. 304–307.
- [15] S.-g. Jung, B. Kang, S. Yeoum, and H. Choo, "Trail-using ant behavior based energy-efficient routing protocol in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 12, no. 4, p. 7350427, 2016.
- [16] N. S. ns3, *Network Simulator*, 2011 (accessed July 2, 2019). [Online]. Available: <https://www.nsnam.org/>
- [17] G. Carneiro, P. Fortuna, and M. Ricardo, "Flowmonitor: a network monitoring framework for the network simulator 3 (ns-3)," in *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*. ICST (Institute for Computer Sciences, Social-Informatics and , 2009, p. 1.
- [18] P. D. S. Kim, *AML2FIG*, 2007 (accessed July 3, 2019). [Online]. Available: <http://www.engr.iupui.edu/~dskim/Distribution/aml2fig/index.php?section=main>
- [19] N. Semiconductors, *MKW41Z/31Z/21Z Data Sheet*, 2018 (accessed July 2, 2019). [Online]. Available: <https://www.nxp.com/docs/en/data-sheet/MKW41Z512.pdf>
- [20] Energizer, *ENERGIZER CR2032*, (accessed July 2, 2019). [Online]. Available: <http://data.energizer.com/pdfs/cr2032.pdf>
- [21] Mathworks, *MATLAB*, 2019 (accessed July 15, 2019). [Online]. Available: <https://www.mathworks.com/products/matlab.html>

## APPENDICES

## A. APPENDIX RSSI VS DISTANCE RELATIONSHIP



---

```

1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2  /*
3   * Copyright (c) 2009 IITP RAS
4   *
5   * This program is free software; you can redistribute it and/or modify
6   * it under the terms of the GNU General Public License version 2 as
7   * published by the Free Software Foundation;
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, write to the Free Software
16  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17  *
18  *
19  * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
20  * Pavel Boyko <boyko@iitp.ru>
21  */
22
23 #ifndef antcomm_DPD_H
24 #define antcomm_DPD_H
25
26 #include "antcomm-id-cache.h"
27 #include "ns3/nstime.h"
28 #include "ns3/packet.h"
29 #include "ns3/ipv4-header.h"
30
31 namespace ns3
32 {
33 namespace antcomm
34 {
35 /**
36  * \ingroup antcomm
37  *
38  * \brief Helper class used to remember already seen packets and detect duplicates.
39  *
40  * Currently duplicate detection is based on unique packet ID given by Packet::GetUId ()
41  * This approach is known to be weak and should be changed.
42  */
43 class DuplicatePacketDetection
44 {
45 public:
46     /// C-tor
47     DuplicatePacketDetection (Time lifetime) : m_idCache (lifetime) {}
48     /// Check that the packet is duplicated. If not, save information about this packet.
49     bool IsDuplicate (Ptr<const Packet> p, const Ipv4Header & header);
50     /// Set duplicate records lifetimes
51     void SetLifetime (Time lifetime);
52     /// Get duplicate records lifetimes
53     Time GetLifetime () const;
54 private:
55     /// Impl
56     IdCache m_idCache;
57 };
58
59 }
60 }
61
62 #endif /* antcomm_DPD_H */

```

---

---

```

1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2  /*
3   * Copyright (c) 2009 IITP RAS
4   *
5   * This program is free software; you can redistribute it and/or modify
6   * it under the terms of the GNU General Public License version 2 as
7   * published by the Free Software Foundation;
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, write to the Free Software
16  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17  *
18  *
19  * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
20  * Pavel Boyko <boyko@iitp.ru>
21  *          Arush Sharma <sharmaas@iupui.edu>
22  */
23
24 #include "antcomm-dpd.h"
25
26 namespace ns3
27 {
28     namespace antcomm
29     {
30
31         bool
32         DuplicatePacketDetection::IsDuplicate (Ptr<const Packet> p, const Ipv4Header & header)
33         {
34             return m_idCache.IsDuplicate (header.GetSource (), p->GetUid ());
35         }
36         void
37         DuplicatePacketDetection::SetLifetime (Time lifetime)
38         {
39             m_idCache.SetLifetime (lifetime);
40         }
41
42         Time
43         DuplicatePacketDetection::GetLifetime () const
44         {
45             return m_idCache.GetLifeTime ();
46         }
47     }
48 }
49 
```

---

---

```

1  /*
2  * Copyright (c) 2015 SKKU Networking Laboratory, IUPUI
3  *
4  * Authors: Soon-gyo Jung <soongyo@skku.edu>, Arush Sharma <sharmaas@iupui.edu>
5  */
6
7  #include "antcomm-helper.h"
8  #include "antcomm-protocol.h"
9  #include "ns3/core-module.h"
10 #include "ns3/network-module.h"
11 #include "ns3/applications-module.h"
12 #include "ns3/mobility-module.h"
13 #include "ns3/config-store-module.h"
14 #include "ns3/wifi-module.h"
15 #include "ns3/internet-module.h"
16 #include "ns3/energy-module.h"
17 #include "ns3/wifi-radio-energy-model-helper.h"
18 #include "ns3/flow-monitor-module.h"
19 #include "ns3/gnuplot.h"
20 #include <iostream>
21 #include <fstream>
22 #include <cmath>
23 #include <sstream>
24 #include "ns3/netanim-module.h"
25 #include <cstdlib>
26 #include <vector>
27 #include <string>
28 #include "ns3/object.h"
29 #include "ns3/uinteger.h"
30 #include "ns3/traced-value.h"
31 #include "ns3/trace-source-accessor.h"
32 #include "ns3/v4ping-helper.h"
33 #include "ns3/packet-sink.h"
34 #include "ns3/trace-helper.h"
35 #include "ns3/snr-tag.h"
36
37
38 using namespace ns3;
39 NS_LOG_COMPONENT_DEFINE ("antcomm");
40
41 class antcommExample {
42 public:
43     antcommExample (uint32_t index, uint32_t size);
44     /// Configure script parameters, \return true on successful configuration
45     bool Configure (int argc, char **argv);
46     /// Run simulation
47     void Run ();
48     /// Report results
49     void Report (std::ostream & os);
50
51 private:
52     /// parameters
53     uint32_t index;
54
55     /// Number of nodes
56     uint32_t size;
57
58     /// Simulation time, seconds
59     double totalTime;
60
61     double TxRange; /// transmission range to set
62
63     /// Write per-device PCAP traces if true
64     bool pcap;
65     /// Print routes if true
66     bool printRoutes;
67
68     uint32_t explorationTime;
69
70     double distanceForTimer;
71
72     uint32_t packetSize; /// bytes
73     /// Convert to time object
74     uint32_t numPackets; /// number of packets to send
75
76     double interval; /// seconds
77
78     std::string topologyFilePath;
79
80

```

```

81 // network
82 NodeContainer nodes;
83 NetDeviceContainer devices;
84 Ipv4InterfaceContainer interfaces;
85 antcommHelper antcomm;
86
87 private:
88 void CreateNodes ();
89 void CreateDevices ();
90 void SetPositions ();
91 void InstallInternetStack ();
92 void SetDistances();
93 void InstallApplications ();
94 void MakeCoordinateFile ();
95 void RandomNodes ();
96 void BackgroundTraffic ();
97 void PrintNodesInformation ();
98 };
99 void GetStatistics(Ptr<FlowMonitor> flowmon, FlowMonitorHelper& flowmonHelper)
100 {
101     flowmon->CheckForLostPackets ();
102     flowmon->SerializeToXmlFile("myproject", true, true);
103     Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmonHelper.GetClassifier ())
104     );
105     std::map<FlowId, FlowMonitor::FlowStats> stats = flowmon->GetFlowStats ();
106
107     uint32_t txPacketsum = 0;
108     uint32_t rxPacketsum = 0;
109     uint32_t DropPacketsum = 0;
110     uint32_t LostPacketsum = 0;
111     double Delaysum = 0;
112
113     std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i;
114     for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end ();
115         ++i)
116     {
117         txPacketsum += i->second.txPackets;
118         rxPacketsum += i->second.rxPackets;
119         LostPacketsum += i->second.lostPackets;
120         DropPacketsum += i->second.packetsDropped.size();
121         Delaysum += i->second.delaySum.GetSeconds();
122
123         std::cout << "Inside_for_loop" << "\n";
124         Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
125
126         std::cout << "Flow_" << i->first << "_" << t.sourceAddress << "-" << t.destinationAddress << "
127         )";
128         std::cout << "Flow_" << i->first << "_" << t.sourceAddress << "-" << t.destinationAddress << "
129         )\n";
130         std::cout << "Tx_Bytes:" << i->second.txBytes << "\n";
131         std::cout << "Rx_Bytes:" << i->second.rxBytes << "\n";
132         std::cout << "Throughput:" << i->second.rxBytes * 8.0 / (i->second.timeLastRxPacket.GetSeconds
133             () - i->second.timeFirstTxPacket.GetSeconds()) << "bps\n";
134         std::cout << "Lost_Packets:" << i->second.lostPackets << "\n";
135         std::cout << "Dropped_Packets:" << i->second.packetsDropped.size() << "\n";
136         std::cout << "Bytes_Dropped:" << i->second.bytesDropped.size() << "\n";
137         std::cout << "Total_received_packets:" << i->second.rxPackets << "\n";
138         std::cout << "Total_transmitted_packets:" << i->second.txPackets << "\n";
139         std::cout << "DelaySum_s:" << i->second.delaySum.GetSeconds () << "\n";
140         std::cout << "Mean_Delay:" << i->second.delaySum.GetSeconds () / i->second.rxPackets << "\n";
141         std::cout << "AverageHopCount:" << 1 + (double)i->second.timesForwarded / (double)i->second.
142             rxPackets << "\n";
143         std::cout << "Packet_Delivery_Ratio:" << i->second.rxPackets * 100 / i->second.txPackets << "%"
144             << "\n";
145         std::cout << "Mean_jitter:" << i->second.jitterSum.GetSeconds () / (i->second.rxPackets - 1)
146             << "\n";
147         std::cout << "Time:" << Simulator::Now().GetSeconds() << "\n";
148         std::cout << "\n\n";
149     }
150
151     std::cout << "All_Tx_Packets:" << txPacketsum << "\n";
152     std::cout << "All_Rx_Packets:" << rxPacketsum << "\n";
153     std::cout << "All_Delay:" << Delaysum / txPacketsum << "\n";
154     std::cout << "All_Lost_Packets:" << LostPacketsum << "\n";
155     std::cout << "All_Drop_Packets:" << DropPacketsum << "\n";
156     std::cout << "Packets_Delivery_Ratio:" << ((rxPacketsum * 100) / txPacketsum) << "%" << "\n";
157     std::cout << "Packets_Lost_Ratio:" << ((LostPacketsum * 100) / txPacketsum) << "%" << "\n";
158 }

```

```

153 void ThroughputMonitor (FlowMonitorHelper *fmhelper, Ptr<FlowMonitor> flowMon)
154 {
155     // p is for previous
156     double localThrou=0;
157     double delay = 0;
158     double avgdelay = 0;
159     uint32_t txPacketsum = 0;
160     static uint32_t ptxPacketsum = 0;
161     uint32_t rxPacketsum = 0;
162     static uint32_t prxPacketsum = 0;
163     uint32_t DropPacketsum = 0;
164     uint32_t LostPacketsum = 0;
165     uint64_t packetforwarded = 0;
166     uint64_t bytes_Dropped = 0;
167     uint64_t rx_Bytes = 0;
168     uint64_t tx_Bytes = 0;
169     double delaysum = 0;
170     static double pdelaysum = 0;
171     double hopcount;
172     static int count = 0;
173
174     std::map<FlowId, FlowMonitor::FlowStats> flowStats = flowMon->GetFlowStats();
175     Ptr<Ipv4FlowClassifier> classing = DynamicCast<Ipv4FlowClassifier> (fmhelper->GetClassifier());
176     for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator stats = flowStats.begin (); stats !=
177         flowStats.end (); ++stats)
178     {
179         Ipv4FlowClassifier::FiveTuple fiveTuple = classing->FindFlow (stats->first);
180         if (fiveTuple.destinationPort == 114/*fiveTuple.sourceAddress == "10.0.0.22" by fiveTuple.
181             destinationAddress == "10.0.0.11"*/ // fiveTuple.destinationPort == 114
182         {
183             if (count > 0)
184             {
185                 // Do something
186                 txPacketsum = stats->second.txPackets /*- ptxPacketsum*/; // modified on 18th
187                     feb 2019
188                 //printf("Current guy %d Previous guy %d\n", txPacketsum, ptxPacketsum);
189                 rxPacketsum = stats->second.rxPackets /*- prxPacketsum*/; // modified on 18th
190                     feb 2019
191                 delaysum = stats->second.delaySum.GetSeconds () - pdelaysum;
192             }
193             else
194             {
195                 txPacketsum = stats->second.txPackets;
196                 rxPacketsum = stats->second.rxPackets;
197                 delaysum = stats->second.delaySum.GetSeconds ();
198             }
199         }
200         ptxPacketsum = stats->second.txPackets;
201         prxPacketsum = stats->second.rxPackets;
202         pdelaysum = stats->second.delaySum.GetSeconds ();
203         LostPacketsum = stats->second.lostPackets;
204         DropPacketsum = stats->second.packetsDropped.size();
205         packetforwarded =stats->second.timesForwarded;
206         localThrou=(stats->second.rxBytes * 8.0 / (stats->second.timeLastRxPacket.GetSeconds()-stats->
207             second.timeFirstTxPacket.GetSeconds()));
208         delay = delaysum /*/ rxPacketsum*/; // modified on 18th feb 2019
209         avgdelay = stats->second.delaySum.GetSeconds() / stats->second.rxPackets;
210         hopcount = 1 + (double)packetforwarded/(double)(stats->second.rxPackets);
211         bytes_Dropped = stats->second.bytesDropped.size();
212         rx_Bytes = stats->second.rxBytes;
213         tx_Bytes = stats->second.txBytes;
214
215         std::stringstream ss;
216         ss << "statistics" << ".csv";
217         static std::ofstream f (ss.str().c_str(), std::ios::out);
218
219         //f << "Throughput\t\t" <<"Delay\t\t\t\t\t" << "AvgDelay\t\t" << "Bytes_Dropped\t\t" << "Tx_Pkt\t\t"
220             << "Rcvd_Pkt\t\t" << "Lst_Pkts\t\t" << "Dpd_Pkts\t\t" << "Hop Count\t\t" << "Rcvd Bytes
221             \t\t" << "Tx Bytes\t\t" << "Time" << std::endl;
222         f <<
223             localThrou << ", " <<
224             delay << ", " <<
225             avgdelay << ", " <<
226             bytes_Dropped << ", " <<
227             txPacketsum << ", " <<
228             rxPacketsum << ", " <<
229             LostPacketsum << ", " <<
230             DropPacketsum << ", " <<
231             hopcount << ", " <<
232             rx_Bytes << ", " <<
233             tx_Bytes << ", " <<

```



```

227         Simulator::Now().GetSeconds() <<
228         std::endl;
229     //break;
230 }
231 }
232 count++;
233 printf("The counter value is %d\n", count);
234 Simulator::Schedule(Seconds(120), &ThroughputMonitor, fmhelper, flowMon); //0.125
235 flowMon->SerializeToXmlFile ("ThroughputMonitor.xml", true, true);
236
237 }
238
239 /// Trace function for remaining energy at node.
240 template <int node>
241 void RemainingEnergyTrace (double oldValue, double newValue)
242 {
243     std::stringstream ss;
244     ss << "remaining_energy_" << node << ".log";
245
246     static std::fstream f (ss.str().c_str(), std::ios::out);
247
248     f << Simulator::Now().GetSeconds() << "_" << newValue << std::endl;
249 }
250
251 /// Trace function for total energy consumption at node.
252 void
253 TotalEnergy (double oldValue, double totalEnergy)
254 {
255     NS_LOG_UNCOND (Simulator::Now ().GetSeconds ()
256         << "s_Total_energy_consumed_by_radio=" << totalEnergy << "J");
257 }
258
259 void EnergyDepletion ()
260 {
261     std::stringstream ss;
262     ss << "energy_" << ".log";
263
264     static std::fstream f (ss.str().c_str(), std::ios::out);
265
266     f << "I am dead at Time " << Simulator::Now().GetSeconds() << std::endl;
267
268     NS_LOG_UNCOND ("I am dead at Time " << Simulator::Now ().GetSeconds () << "s");
269 }
270
271 void TTLchecker(Ptr <const Packet> p, Ptr<Ipv4> ipv4, uint32_t test)
272 {
273     static int i = 0;
274     static double hopsquared = 0;
275     static double hopstotal = 0;
276     Ipv4Header header;
277     p->PeekHeader (header);
278     uint8_t ttl = header.GetTtl ();
279     int hops = 64 - (int)ttl + 1;
280     if (header.GetSource () == Ipv4Address ("10.0.0.2") && header.GetDestination () == Ipv4Address ("
        10.0.0.99"))
281     {
282         i++;
283         hopsquared += hops*hops;
284         hopstotal += hops;
285         std::cout << "Total number of packets received " << i << std::endl;
286         std::stringstream ss;
287         ss << "hopcount" << ".log";
288
289         static std::fstream f (ss.str().c_str(), std::ios::out);
290
291         f << Simulator::Now().GetSeconds () << "_" << "Received Packet with hopCount " << 64-(int)ttl +
            1 << " " << hopstotal << " " << hopsquared << std::endl;
292     }
293 }
294
295 void SendingPacket (Ptr<const Packet> packet)
296 {
297     static int i = 0;
298     std::stringstream ss;
299     ss << "packets_sent" << ".txt";
300
301     static std::fstream f (ss.str().c_str(), std::ios::out);
302     i++;
303
304     f << "At time " << Simulator::Now().GetSeconds () << " packet " << i << " is sent with size "
        << packet->GetSize () << "\n";

```

```

305 }
306
307 void ReceivingPacket (Ptr<const Packet> packet)
308 {
309     static int i = 0;
310     std::stringstream ss;
311     ss << "packets_received" << ".txt";
312
313     static std::fstream f (ss.str().c_str(), std::ios::out);
314     i++;
315
316     f << "At_time_" << Simulator::Now().GetSeconds () << "_packet_" << i << "_is_received_with_"
        size_ << packet->GetSize () << "\n";
317 }
318
319 void ReceivingDrop (Ptr<const Packet> packet)
320 {
321     static int i = 0;
322     std::stringstream ss;
323     ss << "drop_rx" << ".txt";
324
325     static std::fstream f (ss.str().c_str(), std::ios::out);
326     i++;
327
328     f << "At_time_" << Simulator::Now().GetSeconds () << "_packet_" << i << "_is_dropped_while_"
        receiving_with_size_ << packet->GetSize () << "\n";
329 }
330
331 void SendingDrop (Ptr<const Packet> packet)
332 {
333     static int i = 0;
334     std::stringstream ss;
335     ss << "drop_tx" << ".txt";
336
337     static std::fstream f (ss.str().c_str(), std::ios::out);
338     i++;
339
340     f << "At_time_" << Simulator::Now().GetSeconds () << "_packet_" << i << "_is_dropped_while_"
        sending_with_size_ << packet->GetSize () << "\n";
341 }
342
343 void Retransmission (Mac48Address value)
344 {
345     static int i = 0;
346     std::stringstream ss;
347     ss << "transmission" << ".txt";
348
349     static std::fstream f (ss.str().c_str(), std::ios::out);
350     i++;
351     f << i << "Transmission_of_data_packet_has_failed_at_time_" << Simulator::Now().GetSeconds ()
        << "\n";
352 }
353
354 void MaxAttempts (Mac48Address value)
355 {
356     std::stringstream ss;
357     ss << "max_tries" << ".txt";
358
359     static std::fstream f (ss.str().c_str(), std::ios::out);
360     f << "The_transmission_of_a_data_packet_has_exceeded_the_maximum_number_of_attempts_at_time_"
        << Simulator::Now().GetSeconds () << "\n";
361 }
362
363 int main (int argc, char **argv) {
364     uint32_t start = 100;
365     uint32_t index = 1;
366     antcommExample test(index, start);
367     if (!test.Configure (argc, argv))
368         NS_FATAL_ERROR ("Configuration_failed.Aborted.");
369     test.Run ();
370     test.Report (std::cout);
371     return 0;
372 }
373 void ReceivePacket (Ptr<Socket> socket)
374 {
375     while (socket->Recv ())
376     {
377         NS_LOG_UNCOND ("Received_one_packet!");
378     }
379 }
380

```

```

381 static void GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,
382                             uint32_t pktCount, Time pktInterval )
383 {
384     if (pktCount > 0)
385     {
386         //Ptr <UniformRandomVariable> x = CreateObject <UniformRandomVariable> ();
387         //pktSize = x->GetInteger(200, 1400);
388         std::stringstream ss;
389         ss << "packet_size" << ".log";
390
391         static std::fstream f (ss.str().c_str(), std::ios::out);
392
393         //f << Simulator::Now().GetSeconds() << " packet size=" << pktSize << std::endl;
394         socket->Send (Create<Packet> (pktSize));
395         Simulator::Schedule (pktInterval, &GenerateTraffic,
396                             socket, pktSize, pktCount-1, pktInterval);
397     }
398     else
399     {
400         socket->Close ();
401     }
402 }
403
404 //-----
405 antcommExample::antcommExample (uint32_t index, uint32_t size) :
406     index (index),
407     size (size),
408     totalTime (500),
409     TxRange (93),
410     pcap (true),
411     printRoutes (true),
412     explorationTime (50),
413     distanceForTimer(0.5),
414     packetSize(1200),
415     numPackets(500000),
416     interval(0.125){ // 0.16, 0.12, 0.14
417 }
418
419 bool
420 antcommExample::Configure (int argc, char **argv)
421 {
422     // Enable AODV logs by default. Comment this if too noisy
423     //LogComponentEnable("antcommProtocol", LOG_LEVEL_INFO);
424     //LogComponentEnable("antcommRoutingTable", LOG_LEVEL_INFO);
425
426     SeedManager::SetSeed(12345); // 12345
427     SeedManager::SetRun(10);
428
429     CommandLine cmd;
430
431     cmd.AddValue ("pcap", "Write_PCAP_traces.", pcap);
432     cmd.AddValue ("printRoutes", "Print_routing_table_dumps.", printRoutes);
433     cmd.AddValue ("size", "Number_of_nodes.", size);
434     cmd.AddValue ("TxRange", "Transmission_Range_to_set.", TxRange);
435     cmd.AddValue ("totalTime", "Simulation_time_s.", totalTime);
436     cmd.AddValue ("explorationTime", "Exploation_time_s.", explorationTime);
437     cmd.AddValue ("topologyFilePath", "Topology_file_path", topologyFilePath);
438     cmd.AddValue ("distanceForTimer", "The_distance_for_timer", distanceForTimer);
439     cmd.AddValue ("packetSize", "The_packet_size", packetSize);
440     cmd.AddValue ("numPackets", "The_number_of_packets", numPackets);
441     cmd.Parse (argc, argv);
442
443     std::ostringstream s;
444     s << "output/topology_case181_" << index << "_" << size << ".csv";
445     topologyFilePath = s.str();
446     return true;
447 }
448
449 void antcommExample::Run () {
450     Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", UIntegerValue (2200)); //
451         enable rts cts all the time.
452
453     CreateNodes ();
454     SetPositions();
455     CreateDevices ();
456     BasicEnergySourceHelper basicSourceHelper;
457     // I am changing the battery capacity in order to make the simulation run for full 500 seconds

```

```

459 basicSourceHelper.Set ("BasicEnergySourceInitialEnergyJ", DoubleValue (0.4)); // 470 1102. 1270 is
    for 120 meters and 1237 for 99 meters
460
461 EnergySourceContainer sources = basicSourceHelper.Install (nodes);
462
463 WifiRadioEnergyModelHelper radioEnergyHelper;
464
465 radioEnergyHelper.Set ("TxCurrentA", DoubleValue (0.0061)); //0.0061 and 0.4 joules as initial
    energy
466 radioEnergyHelper.Set ("RxCurrentA", DoubleValue (0.0068)); //0.0068
467 radioEnergyHelper.Set ("IdleCurrentA", DoubleValue (0.00019)); //0.00019
468
469 radioEnergyHelper.SetDepletionCallback (MakeCallback (&EnergyDepletion));
470 DeviceEnergyModelContainer deviceModels = radioEnergyHelper.Install (devices, sources);
471 InstallInternetStack ();
472 MakeCoordinateFile ();
473 Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/ns3::WifiNetDevice/RemoteStationManager/
    MacTxDataFailed", MakeCallback (&Retransmission));
474 Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/ns3::WifiNetDevice/RemoteStationManager/
    MacTxFinalDataFailed", MakeCallback (&MaxAttempts));
475 Config::ConnectWithoutContext ("/NodeList/98/ns3::Ipv4L3Protocol/Rx", MakeCallback(&TTLchecker));//
    99
476 Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/ns3::WifiNetDevice/Mac/MacTx",
    MakeCallback (&SendingPacket));
477 Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/ns3::WifiNetDevice/Mac/MacRx",
    MakeCallback (&ReceivingPacket));
478 Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/ns3::WifiNetDevice/Mac/MacRxDrop",
    MakeCallback (&ReceivingDrop));
479 Config::ConnectWithoutContext ("/NodeList/*/DeviceList/*/ns3::WifiNetDevice/Mac/MacTxDrop",
    MakeCallback (&SendingDrop));
480 //Simulator::Schedule (Seconds(50), &antcommExample::RandomNodes, this);
481 //Simulator::Schedule (Seconds(100), &antcommExample::RandomNodes, this);
482 //Simulator::Schedule (Seconds(150), &antcommExample::RandomNodes, this);
483 //Simulator::Schedule (Seconds(200), &antcommExample::RandomNodes, this);
484 //RandomNodes ();
485 InstallApplications ();
486 /*for (uint32_t i = 0 ; i < size; i++)
487 {
488     // energy source
489     Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (sources.Get (i));
490     basicSourcePtr->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback (&RemainingEnergy));
491
492     // device energy model
493     Ptr<DeviceEnergyModel> basicRadioModelPtr = basicSourcePtr->FindDeviceEnergyModels ("ns3::
    WifiRadioEnergyModel").Get (0);
494     NS_ASSERT (basicRadioModelPtr != NULL);
495     basicRadioModelPtr->TraceConnectWithoutContext ("TotalEnergyConsumption", MakeCallback (&
    TotalEnergy));
496
497 }*/
498 // Tried the loop way, but it doesn't work. If there's another solution to it, that would be good
499 sources.Get (0)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <0>));
500 sources.Get (1)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <1>));
501 sources.Get (2)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <2>));
502 sources.Get (3)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <3>));
503 sources.Get (4)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <4>));
504 sources.Get (5)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <5>));
505 sources.Get (6)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <6>));
506 sources.Get (7)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <7>));
507 sources.Get (8)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <8>));
508 sources.Get (9)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <9>));
509 sources.Get (10)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <10>));
510 sources.Get (11)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <11>));
511 sources.Get (12)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <12>));
512 sources.Get (13)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
    <13>));

```

[illegible]

[illegible]

```

593 sources.Get (94)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
594 <94>));
594 sources.Get (95)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
595 <95>));
595 sources.Get (96)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
596 <96>));
596 sources.Get (97)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
597 <97>));
597 sources.Get (98)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
598 <98>));
598 sources.Get (99)->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback(&RemainingEnergyTrace
599 <99>));
600
601 std::cout << "Starting simulation antcomm case181 No. " << index << " for " << totalTime << " s...\n";
602
603 //std::ostream s;
604 //s << "animation_" << index << "_" << size << ".xml";
605 /*Ptr<FlowMonitor> flowMonitor;
606 FlowMonitorHelper flowHelper;
607 flowMonitor = flowHelper.InstallAll();*/
608 FlowMonitorHelper* flowmonHelper = new FlowMonitorHelper();
609 Ptr<FlowMonitor> flowmon = flowmonHelper->InstallAll ();
610
611 ThroughputMonitor(flowmonHelper, flowmon); // just changed this
612
613 Simulator::Stop (Seconds (totalTime));
614 //AnimationInterface anim (s.str());
615 AnimationInterface anim ("animation.xml");
616 anim.UpdateNodeColor(nodes.Get(1), 0, 0, 255);
617 anim.UpdateNodeColor(nodes.Get(98), 0, 0, 255);
618 Simulator::Run ();
619 /*for (DeviceEnergyModelContainer::Iterator iter = deviceModels.Begin (); iter != deviceModels.End ()
620 ; iter++)
621 {
622     double energyConsumed = (*iter)->GetTotalEnergyConsumption ();
623     NS_LOG_UNCOND ("End of simulation (" << Simulator::Now ().GetSeconds () << "s) Total energy
624     consumed by radio = " << energyConsumed << "J");
625 }*/
626 PrintNodesInformation();
627 antcommHelper::EmptyAllInformation();
628 //ThroughputMonitor(flowmonHelper, flowmon);
629 GetStatistics(flowmon,*flowmonHelper);
630 Simulator::Destroy ();
631 }
632
633 void antcommExample::Report (std::ostream &) {
634 }
635
636 void antcommExample::CreateNodes () {
637     NS_LOG_UNCOND ("creating the nodes");
638     // NodeContainer sinkNode;
639     // NodeContainer sensorNodes;
640     std::cout << "Creating " << (unsigned)size << " nodes.\n";
641     // sinkNode.Create(1);
642     // sensorNodes.Create(size);
643
644     nodes.Create(size);
645
646     //Name sink
647     /* std::ostream os;
648     os << "sink";
649     Names::Add (os.str (), nodes.Get (0));
650
651     // Name sensor nodes
652     for (uint32_t i = 1; i < size; ++i) {
653         std::ostream os;
654         os << "node-" << i;
655         Names::Add (os.str (), nodes.Get (i));
656     }
657
658     // nodes = NodeContainer(sinkNode, sensorNodes); */
659 }
660
661 void antcommExample::CreateDevices () {
662     NS_LOG_UNCOND ("setting the default phy and channel parameters");
663     //double ttxp = 87; // 87 // 60
664     WifiMacHelper wifiMac;
665     wifiMac.SetType ("ns3::AdhocWifiMac");

```

```

664 YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
665 //wifiPhy.Set ("TxPowerStart",DoubleValue (txp));
666 //wifiPhy.Set ("TxPowerEnd",DoubleValue (150)); // 95
667 //wifiPhy.Set ("TxPowerLevels", UIntegerValue (60));
668 //wifiPhy.Set ("ChannelWidth", UIntegerValue (160));
669 YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
670 //wifiChannel.AddPropagationLoss ("ns3::FrisPropagationLossModel");
671 /*For the current topology, 63 meter is minimum transmission range and 71 meter for background
   traffic*/
672 wifiChannel.AddPropagationLoss ("ns3::RangePropagationLossModel", "MaxRange", DoubleValue(TxRange));
673 //15 22
674 wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
675 wifiPhy.SetChannel (wifiChannel.Create ());
676 WifiHelper wifi;
677 //wifi.SetStandard (WIFI_PHY_STANDARD_80211g);
678 wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode", StringValue ("
   OfdmRate6Mbps")); // OfdmRate6Mbps
679 devices = wifi.Install (wifiPhy, wifiMac, nodes);
680 if (pcap) {
681     wifiPhy.EnablePcapAll (std::string ("antcomm"));
682 }
683
684 void antcommExample::SetPositions() {
685
686     NS_LOG_UNCOND ("Assigning_mobility");
687     MobilityHelper mobility;
688     int px, py/*, dpx, dpy*/;
689     Ptr<ListPositionAllocator> node_coordinates = CreateObject<ListPositionAllocator> ();
690     Ptr<NormalRandomVariable> x = CreateObject<NormalRandomVariable> ();
691     x->SetAttribute ("Variance", DoubleValue (99));
692     FILE *stream/*, *stream2*/;
693     stream = fopen("/export/home1/sharmaas/newworkspace/ns-allinone-3.25/ns-3.25/scratch/antcomm/
   cellular_layout.txt", "r"); // grid_layout
694     //stream2 = fopen("/export/home1/sharmaas/newworkspace/ns-allinone-3.25/ns-3.25/scratch/antcomm/
   cellular_layout.txt", "w");
695     if (stream == NULL)
696         printf ("FILE_not_found\n");
697     for(uint32_t i = 0; i < size; i++)
698     {
699         fscanf(stream, "%d_%d", &px, &py);
700         //dpx = px + x->GetInteger ();
701         //dpy = py + x->GetInteger ();
702         //fprintf(stream2, "%d %d\n", dpx, dpy);
703         node_coordinates->Add (Vector (px, py, 0.0));
704     }
705     fclose(stream);
706     //fclose(stream2);
707     mobility.SetPositionAllocator(node_coordinates);
708     mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
709     /*double distance = 50;
710     mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
711         "MinX", DoubleValue (0.0),
712         "MinY", DoubleValue (0.0),
713         "DeltaX", DoubleValue (distance),
714         "DeltaY", DoubleValue (distance),
715         "GridWidth", UIntegerValue (10),
716         "LayoutType", StringValue ("RowFirst"));*/
717     mobility.Install (nodes);
718 }
719
720
721 void antcommExample::PrintNodesInformation(){
722
723     std::ostringstream s;
724     s << "antcomm.nodes.information_case181_" << index << "_" << size;
725     Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> (s.str(), std::ios::out);
726     antcomm.PrintNodesInformation(routingStream);
727 }
728
729 void antcommExample::InstallInternetStack () {
730     // you can configure AODV attributes here using aodv.Set(name, value)
731     //antree.Set("m_sink", );
732     //antree.Set("TransmissionRange", UIntegerValue (55));
733     antcommHelper antcomm;
734     InternetStackHelper stack;
735     antcomm.Set("DistanceForTimer", DoubleValue(distanceForTimer));
736     antcomm.Set("Alpha", DoubleValue(0.5));
737     antcomm.Set("Beta", DoubleValue(0.8));

```



```

739 antcomm.Set("Gamma", DoubleValue(0.8));
740 stack.SetRoutingHelper (antcomm); // has effect on the next Install ()
741 stack.Install (nodes);
742 Ipv4AddressHelper address;
743 address.SetBase ("10.0.0.0", "255.0.0.0");
744 interfaces = address.Assign (devices);
745
746 if (printRoutes) {
747     Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("antree.routes", std::ios::
        out);
748     antcomm.PrintRoutingTableAllAt (Seconds (totalTime), routingStream);
749 }
750 }
751
752 void antcommExample::RandomNodes ()
753 {
754     Ptr<UniformRandomVariable> x = CreateObject<UniformRandomVariable> ();
755     x->SetAttribute ("Min", DoubleValue (0));
756     x->SetAttribute ("Max", DoubleValue (size-1));
757     uint32_t sourceNode = x->GetInteger();
758     uint32_t sinkNode = x->GetInteger();
759     while (sourceNode==sinkNode || sourceNode==98 || sourceNode==1 || sinkNode==98 || sinkNode==1)
760     {
761         sourceNode = x->GetInteger();
762         sinkNode = x->GetInteger();
763     }
764     UdpServerHelper myServer (14);
765     ApplicationContainer serverApp = myServer.Install (nodes.Get (sinkNode));
766     serverApp.Start (Seconds (1.0));
767     serverApp.Stop (Seconds (totalTime));
768
769     UdpClientHelper client (interfaces.GetAddress (sinkNode), 14);
770     client.SetAttribute ("MaxPackets", UIntegerValue (14000000));
771     client.SetAttribute ("Interval", TimeValue (Time ("0.125"))); // 0.125 packets/s
772     client.SetAttribute ("PacketSize", UIntegerValue (1500)); //bytes
773
774     ApplicationContainer clientApp = client.Install (nodes.Get (sourceNode));
775     clientApp.Start (Seconds (4.0)); //3 4
776     clientApp.Stop (Seconds (totalTime));
777     /*Time interPacketInterval = Seconds (interval);
778     TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
779     Ptr<Socket> recuSink = Socket::CreateSocket (nodes.Get (sinkNode), tid);
780     InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 14);
781     recuSink->Bind (local);
782     recuSink->SetRecvCallback (MakeCallback (&ReceivePacket));
783
784     Ptr<Socket> source = Socket::CreateSocket (nodes.Get (sourceNode), tid);
785     InetSocketAddress remote = InetSocketAddress (interfaces.GetAddress (sinkNode, 0), 14);
786     source->Connect (remote);
787     Simulator::Schedule (Seconds (5.0), &GenerateTraffic, source, packetSize, numPackets,
        interPacketInterval);*/
788
789     //Simulator::Schedule(Seconds(50), &antcommExample::RandomNodes, this);
790 }
791
792 void antcommExample::InstallApplications () {
793
794     NS_LOG_UNCOND ("Traffic_generator_for_data_packet_generation");
795     UdpServerHelper myServer (114);
796     ApplicationContainer serverApp = myServer.Install (nodes.Get (98)); //96
797     serverApp.Start (Seconds (1.0));
798     serverApp.Stop (Seconds (totalTime));
799
800     UdpClientHelper client (interfaces.GetAddress (98), 114);
801     client.SetAttribute ("MaxPackets", UIntegerValue (14000000));
802     client.SetAttribute ("Interval", TimeValue (Time ("120"))); // 0.125 0.08 packets/s
803     client.SetAttribute ("PacketSize", UIntegerValue (1500)); //bytes
804
805     ApplicationContainer clientApp = client.Install (nodes.Get (1)); //4 or 9
806     clientApp.Start (Seconds (3.0)); //3
807     clientApp.Stop (Seconds (totalTime));
808     /*uint32_t sourceNode = 1;
809     uint32_t sinkNode = 98;
810     Time interPacketInterval = Seconds (interval);
811     TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
812     Ptr<Socket> recuSink = Socket::CreateSocket (nodes.Get (sinkNode), tid);
813     InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 114);
814     recuSink->Bind (local);
815     recuSink->SetRecvCallback (MakeCallback (&ReceivePacket));
816

```

```

817  Ptr<Socket> source = Socket::CreateSocket (nodes.Get (sourceNode), tid);
818  InetSocketAddress remote = InetSocketAddress (interfaces.GetAddress (sinkNode, 0), 114);
819  source->Connect (remote);
820  Simulator::Schedule (Seconds (3.0), &GenerateTraffic, source, packetSize, numPackets,
      interPacketInterval);*/
821 }
822 void antcommExample::MakeCoordinateFile ()
823 {
824     FILE *fp;
825     fp = fopen ("id2coordinates.txt", "w");
826     uint32_t i;
827     for (i=0; i < size; i++)
828     {
829         uint32_t id, x1, y1;
830         Ptr <Node> node = nodes.Get (i);
831         id = node->GetId ();
832         Ptr <MobilityModel> mob = node->GetObject<MobilityModel> ();
833         Vector position = mob->GetPosition ();
834         x1 = position.x;
835         y1 = position.y;
836         fprintf (fp, "%d\t%d\t%d\n", id, x1, y1);
837     }
838     fclose (fp);
839 }

```

---

---

```

1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2
3  #include "antcomm-helper.h"
4  #include "antcomm-protocol.h"
5  #include "ns3/core-module.h"
6  #include "ns3/node-list.h"
7  #include "ns3/names.h"
8  #include "ns3/ptr.h"
9  #include "ns3/ipv4-list-routing.h"
10
11 namespace ns3 {
12
13     // std::map<std::string, double> antcommHelper::m_distances;
14     // Line commented out by me
15
16     // std::map<uint16_t, uint16_t> antcommHelper::m_hopCount;
17     // Line commented out by me
18
19     // std::map<uint16_t, double> antcommHelper::m_energy;
20     // Line commented out by me
21
22     // std::map<uint16_t, double> antcommHelper::m_pheromone;
23     // Line commented out by me
24
25     // std::map<uint16_t, double> antcommHelper::m_pheromoneUpdatedTime; // Line commented out by me
26
27     // std::map<uint32_t, Time > antcommHelper::m_startTimestamps; // Line commented out by me
28
29     uint32_t antcommHelper::dataPacketID = 0;
30
31
32     antcommHelper::antcommHelper() : Ipv4RoutingHelper () {
33         m_agentFactory.SetTypeId ("ns3::antcomm::antcommProtocol");
34     }
35
36     antcommHelper* antcommHelper::Copy (void) const {
37         return new antcommHelper (*this);
38     }
39
40     Ptr<Ipv4RoutingProtocol> antcommHelper::Create (Ptr<Node> node) const {
41         Ptr<antcomm::antcommProtocol> agent = m_agentFactory.Create<antcomm::antcommProtocol> ();
42         node->AggregateObject (agent);
43         return agent;
44     }
45
46     void antcommHelper::Set (std::string name, const AttributeValue &value) {
47         m_agentFactory.Set (name, value);
48     }
49
50
51     int64_t antcommHelper::AssignStreams (NodeContainer c, int64_t stream) {
52         int64_t currentStream = stream;
53         Ptr<Node> node;
54         for (NodeContainer::Iterator i = c.Begin (); i != c.End (); ++i) {
55             node = (*i);
56             Ptr<Ipv4> ipv4 = node->GetObject<Ipv4> ();
57             NS_ASSERT_MSG (ipv4, "Ipv4 not installed on node");
58             Ptr<Ipv4RoutingProtocol> proto = ipv4->GetRoutingProtocol ();
59             NS_ASSERT_MSG (proto, "Ipv4 routing not installed on node");
60             Ptr<antcomm::antcommProtocol> antcomm = DynamicCast<antcomm::antcommProtocol> (proto);
61             if (antcomm) {
62                 currentStream += antcomm->AssignStreams (currentStream);
63                 continue;
64             }
65             // antcomm may also be in a list
66             Ptr<Ipv4ListRouting> list = DynamicCast<Ipv4ListRouting> (proto);
67             if (list) {
68                 int16_t priority;
69                 Ptr<Ipv4RoutingProtocol> listProto;
70                 Ptr<antcomm::antcommProtocol> listantcomm;
71                 for (uint32_t i = 0; i < list->GetNRoutingProtocols (); i++) {
72                     listProto = list->GetRoutingProtocol (i, priority);
73                     listantcomm = DynamicCast<antcomm::antcommProtocol> (listProto);
74                     if (listantcomm) {
75                         currentStream += listantcomm->AssignStreams (currentStream);
76                         break;
77                     }
78                 }
79             }
80         }
81     }

```

```

81     return (currentStream - stream);
82 }
83
84 void antcommHelper::PrintNodesInformation(Ptr<OutputStreamWrapper> stream) {
85
86     *stream->GetStream () << "ID\tHopCount\tPheromone\tEnergy\tRecvPacketCount\tHopDistance\t
      tEndToEndDelay\n";
87     for (uint32_t i = 0; i < NodeList::GetNNodes (); i++) {
88         Ptr<Node> node = NodeList::GetNode (i);
89         Ptr<Ipv4> ipv4 = node->GetObject<Ipv4> ();
90         Ptr<Ipv4RoutingProtocol> proto = ipv4->GetRoutingProtocol ();
91         Ptr<antcomm::antcommProtocol> antcomm = DynamicCast<antcomm::antcommProtocol> (proto);
92         if (antcomm) {
93             antcomm->PrintInformation(stream);
94         }
95     }
96 }
97
98
99
100 }

```

---

---

```

1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2  #ifndef antcomm_HELPER_H
3  #define antcomm_HELPER_H
4
5  #include "ns3/object-factory.h"
6  #include "ns3/node.h"
7  #include "ns3/node-container.h"
8  #include "ns3/ipv4-routing-helper.h"
9  #include "ns3/wifi-module.h"
10 #include "ns3/energy-module.h"
11
12 namespace ns3 {
13
14     /**
15      * \ingroup antcomm
16      * \brief Helper class that adds antcomm routing to nodes.
17      */
18     class antcommHelper : public Ipv4RoutingHelper {
19     public:
20         antcommHelper();
21
22         /**
23          * \returns pointer to clone of this OlsrHelper
24          */
25         /**
26          * \internal
27          * This method is mainly for internal use by the other helpers;
28          * clients are expected to free the dynamic memory allocated by this method
29          */
30         antcommHelper* Copy (void) const;
31
32         /**
33          * \param node the node on which the routing protocol will run
34          * \returns a newly-created routing protocol
35          */
36         /**
37          * This method will be called by ns3::InternetStackHelper::Install
38          */
39         /**
40          * \todo support installing AODV on the subset of all available IP interfaces
41          */
42         virtual Ptr<Ipv4RoutingProtocol> Create (Ptr<Node> node) const;
43
44         /**
45          * \param name the name of the attribute to set
46          * \param value the value of the attribute to set.
47          */
48         /**
49          * This method controls the attributes of ns3::aodv::RoutingProtocol
50          */
51         void Set (std::string name, const AttributeValue &value);
52
53         /**
54          * Assign a fixed random variable stream number to the random variables
55          * used by this model. Return the number of streams (possibly zero) that
56          * have been assigned. The Install() method of the InternetStackHelper
57          * should have previously been called by the user.
58          */
59         /**
60          * \param stream first stream index to use
61          * \param c NodeContainer of the set of nodes for which AODV
62          * should be modified to use a fixed stream
63          * \return the number of stream indices assigned by this helper
64          */
65         int64_t AssignStreams (NodeContainer c, int64_t stream);
66
67         void PrintNodesInformation (Ptr<OutputStreamWrapper> stream);
68
69         static void PutDistance(uint16_t sourceID, uint16_t destinationID, double distance){
70
71             std::stringstream buffer;
72             buffer << sourceID << "|" << destinationID;
73             antcommHelper::m_distances[buffer.str()] = distance; // Line commented
74             out by me
75         }
76
77         static void PutPheromone(uint16_t nodeID, double pheromone){
78             //antcommHelper::m_pheromone[nodeID] = pheromone;
79             PutPheromoneUpdatedTime(nodeID, Simulator::Now ());
80         }
81
82         static double GetPheromone(Ipv4Address dst/*uint16_t nodeID*/){
83
84             //std::map<uint16_t,double>::iterator result = antcommHelper::m_pheromone.
85             find(nodeID);
86         }
87     };
88

```

```

79             //if (result != antcommHelper::m_pheromone.end()){
80                 Time offset = Simulator::Now() - Time(GetPheromoneUpdatedTime(dst
                        /*nodeID*/));
81                 //return exp( (-1)*offset.GetSeconds()*0.5 ) * result->second;
82             }else{
83                 //return -1;
84             }
85         return 0; //Line inserted by me
86     }
87     //}
88     static void PutEnergy(uint16_t nodeID, double energy){
89         // antcommHelper::m_energy[nodeID] = energy; //Line commented out by me
90         //BasicEnergySourceHelper basicSourceHelper;
91         //basicSourceHelper.Set ("BasicEnergySourceInitialEnergyJ", DoubleValue (2.0));
92         //EnergySourceContainer sources = basicSourceHelper.Install (nodes);
93         //Device Energy Model
94         //WifiRadioEnergyModelHelper radioEnergyHelper;
95         //radioEnergyHelper.Set ("TxCurrentA", DoubleValue (0.0174));
96         //DeviceEnergyModelContainer deviceModels = radioEnergyHelper.Install (devices, sources);
97     }
98
99     static double GetEnergy(Ipv4Address dst/*uint16_t nodeID*/){
100         /*std::map<uint16_t,double>::iterator result = antcommHelper::m_energy.
101             find(nodeID);
102         if (result != antcommHelper::m_energy.end()){
103             return result->second;
104         }else{
105             return -1;
106         } */
107     }
108     return 0;
109 }
110
111 static void PutPheromoneUpdatedTime(uint16_t nodeID, Time time){
112     // antcommHelper::m_pheromoneUpdatedTime[nodeID] = time.GetDouble();
113 }
114
115 static double GetPheromoneUpdatedTime(Ipv4Address dst/*uint16_t nodeID*/){
116     /*std::map<uint16_t,double>::iterator result = antcommHelper::
117         m_pheromoneUpdatedTime.find(nodeID);
118     if (result != antcommHelper::m_pheromoneUpdatedTime.end()){
119         return result->second;
120     }
121     }
122     else{
123         return -1;
124     } */
125 }
126 return 0;
127 }
128
129 static void PutHopCount(uint16_t nodeID, uint16_t hopCount){
130     // antcommHelper::m_hopCount[nodeID] = hopCount; //Line commented out by me
131 }
132
133 static uint16_t GetHopCount(Ipv4Address dst/*uint16_t nodeID*/){
134     /*std::map<uint16_t,uint16_t>::iterator result = antcommHelper::m_hopCount
135         .find(nodeID);
136     if (result != antcommHelper::m_hopCount.end()){
137         return result->second;
138     }
139     }
140     else{
141         return -1;
142     } */
143 }
144 return 0; //Line inserted by me as its a non void function
145 }
146
147 static void EmptyAllInformation(){
148     // m_distances.clear(); //Line commented out by me
149     // m_hopCount.clear(); //Line commented out by me
150     // m_energy.clear(); //Line commented out by me
151     // m_pheromone.clear(); //Line commented out by me
152     // m_pheromoneUpdatedTime.clear(); //Line commented out by me
153 }
154

```

```

155         static uint32_t PutStartTimestamp(){
156             m_startTimestamps[dataPacketID] = Simulator::Now();
157             return dataPacketID++;
158         }
159
160         static double GetEndToEndDelay(uint32_t packetID){
161             //std::map<uint32_t,Time>::iterator result = m_startTimestamps.find(
162                 packetID);
163             //return Simulator::Now().GetMilliSeconds() - result->second.
164                 GetMilliSeconds() ;
165
166     return 0;
167     }
168
169     private:
170         /** the factory to create antcomm routing object */
171         ObjectFactory m_agentFactory;
172
173         static std::map<std::string, double> m_distances;
174
175         static std::map<uint16_t, uint16_t> m_hopCount;
176
177         static std::map<uint16_t, double> m_energy;
178
179         static std::map<uint16_t, double> m_pheromone;
180
181         static std::map<uint16_t, double> m_pheromoneUpdatedTime;
182
183         static std::map<uint32_t, Time > m_startTimestamps;
184
185         static uint32_t dataPacketID;
186     };
187
188 }
189 #endif /* antcomm_HELPER_H */

```

---

---

```

1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2  /*
3   * Copyright (c) 2009 IITP RAS
4   *
5   * This program is free software; you can redistribute it and/or modify
6   * it under the terms of the GNU General Public License version 2 as
7   * published by the Free Software Foundation;
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, write to the Free Software
16  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17  *
18  * Based on
19  * NS-2 antcomm model developed by the CMU/MONARCH group and optimized and
20  * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
21  *
22  * antcomm-UU implementation by Erik Nordström of Uppsala University
23  * http://core.it.uu.se/core/index.php/antcomm-UU
24  *
25  * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
26  * Pavel Boyko <boyko@iitp.ru>
27  */
28 #include "antcomm-id-cache.h"
29 #include <algorithm>
30
31 namespace ns3
32 {
33 namespace antcomm
34 {
35 bool
36 IdCache::IsDuplicate (Ipv4Address addr, uint32_t id)
37 {
38     Purge ();
39     for (std::vector<UniqueId>::const_iterator i = m_idCache.begin ();
40          i != m_idCache.end (); ++i)
41         if (i->m_context == addr && i->m_id == id)
42             return true;
43     struct UniqueId uniqueId =
44     { addr, id, m_lifetime + Simulator::Now () };
45     m_idCache.push_back (uniqueId);
46     return false;
47 }
48
49 void
50 IdCache::Purge ()
51 {
52     m_idCache.erase (remove_if (m_idCache.begin (), m_idCache.end (),
53                                IsExpired ()), m_idCache.end ());
54 }
55
56 uint32_t
57 IdCache::GetSize ()
58 {
59     Purge ();
60     return m_idCache.size ();
61 }
62
63 }
64
65 }

```

---



---

```

1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2  /*
3   * Copyright (c) 2009 IITP RAS
4   *
5   * This program is free software; you can redistribute it and/or modify
6   * it under the terms of the GNU General Public License version 2 as
7   * published by the Free Software Foundation;
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, write to the Free Software
16  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17  *
18  * Based on
19  * NS-2 antcomm model developed by the CMU/MONARCH group and optimized and
20  * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
21  *
22  * antcomm-UU implementation by Erik Nordström of Uppsala University
23  * http://core.it.uu.se/core/index.php/antcomm-UU
24  *
25  * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
26  * Pavel Boyko <boyko@iitp.ru>
27  */
28
29 #ifndef antcomm_ID_CACHE_H
30 #define antcomm_ID_CACHE_H
31
32 #include "ns3/ipv4-address.h"
33 #include "ns3/simulator.h"
34 #include <vector>
35
36 namespace ns3
37 {
38     namespace antcomm
39     {
40         /**
41          * \ingroup antcomm
42          *
43          * \brief Unique packets identification cache used for simple duplicate detection.
44          */
45         class IdCache
46         {
47         public:
48             /// c-tor
49             IdCache (Time lifetime) : m_lifetime (lifetime) {}
50             /// Check that entry (addr, id) exists in cache. Add entry, if it doesn't exist.
51             bool IsDuplicate (Ipv4Address addr, uint32_t id);
52             /// Remove all expired entries
53             void Purge ();
54             /// Return number of entries in cache
55             uint32_t GetSize ();
56             /// Set lifetime for future added entries.
57             void SetLifetime (Time lifetime) { m_lifetime = lifetime; }
58             /// Return lifetime for existing entries in cache
59             Time GetLifeTime () const { return m_lifetime; }
60         private:
61             /// Unique packet ID
62             struct UniqueId
63             {
64                 /// ID is supposed to be unique in single address context (e.g. sender address)
65                 Ipv4Address m_context;
66                 /// The id
67                 uint32_t m_id;
68                 /// When record will expire
69                 Time m_expire;
70             };
71             struct IsExpired
72             {
73                 bool operator() (const struct UniqueId & u) const
74                 {
75                     return (u.m_expire < Simulator::Now ());
76                 }
77             };
78             /// Already seen IDs
79             std::vector<UniqueId> m_idCache;
80             /// Default lifetime for ID records

```

```
81     Time m_lifetime;  
82 };  
83  
84 }  
85 }  
86 #endif /* antcomm_ID_CACHE_H */
```

---

---

```

1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2  /*
3   * Copyright (c) 2009 IITP RAS
4   *
5   * This program is free software; you can redistribute it and/or modify
6   * it under the terms of the GNU General Public License version 2 as
7   * published by the Free Software Foundation;
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, write to the Free Software
16  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17  *
18  * Based on
19  * NS-2 antcomm model developed by the CMU/MONARCH group and optimized and
20  * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
21  *
22  * antcomm-UU implementation by Erik Nordström of Uppsala University
23  * http://core.it.uu.se/core/index.php/antcomm-UU
24  *
25  * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
26  * Pavel Boyko <boyko@iitp.ru>
27  */
28
29 #include "antcomm-neighbor.h"
30 #include "ns3/log.h"
31 #include <algorithm>
32
33
34 namespace ns3
35 {
36
37   NS_LOG_COMPONENT_DEFINE ("antcommNeighbors");
38
39   namespace antcomm
40   {
41     Neighbors::Neighbors (Time delay) :
42       m_ntimer (Timer::CANCEL_ON_DESTROY)
43     {
44       m_ntimer.SetDelay (delay);
45       m_ntimer.SetFunction (&Neighbors::Purge, this);
46       m_txErrorCallback = MakeCallback (&Neighbors::ProcessTxError, this);
47     }
48
49     bool
50     Neighbors::IsNeighbor (Ipv4Address addr)
51     {
52       Purge ();
53       for (std::vector<Neighbor>::const_iterator i = m_nb.begin ();
54            i != m_nb.end (); ++i)
55       {
56         if (i->m_neighborAddress == addr)
57           return true;
58       }
59       return false;
60     }
61
62     Time
63     Neighbors::GetExpireTime (Ipv4Address addr)
64     {
65       Purge ();
66       for (std::vector<Neighbor>::const_iterator i = m_nb.begin (); i
67            != m_nb.end (); ++i)
68       {
69         if (i->m_neighborAddress == addr)
70           return (i->m_expireTime - Simulator::Now ());
71       }
72       return Seconds (0);
73     }
74
75     void
76     Neighbors::Update (Ipv4Address addr, double snr, double energy, Time expire)
77     {
78       for (std::vector<Neighbor>::iterator i = m_nb.begin (); i != m_nb.end (); ++i)
79         if (i->m_neighborAddress == addr)
80           {

```

```

81         i->m_expireTime
82         = std::max (expire + Simulator::Now (), i->m_expireTime);
83         if (i->m_hardwareAddress == Mac48Address ())
84             i->m_hardwareAddress = LookupMacAddress (i->m_neighborAddress);
85         return;
86     }
87
88     NS_LOG_LOGIC ("Open_link_to_" << addr);
89     Neighbor neighbor (addr, LookupMacAddress (addr), snr, energy, expire + Simulator::Now ());
90     m_nb.push_back (neighbor);
91     Purge ();
92 }
93
94 int Neighbors::Degree ()
95 {
96     return m_nb.size();
97 }
98
99 Ipv4Address Neighbors::GetAddress (int i)
100 {
101     return m_nb[i].m_neighborAddress;
102 }
103
104 double Neighbors::GetEnergy (Ipv4Address addr)
105 {
106     for (std::vector<Neighbor>::iterator i = m_nb.begin (); i != m_nb.end (); ++i)
107     {
108         if (i->m_neighborAddress == addr)
109             return i->residualEnergy;
110     }
111     NS_LOG_LOGIC ("Neighbor_not_found_yet_so_return_zero_energy");
112     return 0;
113 }
114
115 double Neighbors::GetRssi (Ipv4Address addr)
116 {
117     for (std::vector<Neighbor>::iterator i = m_nb.begin (); i != m_nb.end (); ++i)
118     {
119         if (i->m_neighborAddress == addr)
120             return i->signalnoiseratio;
121     }
122     NS_LOG_LOGIC ("Neighbor_not_found_yet_so_RSSI_would_be_0");
123     return 0;
124 }
125
126 struct CloseNeighbor
127 {
128     bool operator() (const Neighbors::Neighbor & nb) const
129     {
130         return ((nb.m_expireTime < Simulator::Now ()) || nb.close);
131     }
132 };
133
134 void
135 Neighbors::Purge ()
136 {
137     if (m_nb.empty ())
138         return;
139
140     CloseNeighbor pred;
141     if (!m_handleLinkFailure.IsNull ())
142     {
143         for (std::vector<Neighbor>::iterator j = m_nb.begin (); j != m_nb.end (); ++j)
144         {
145             if (pred (*j))
146             {
147                 NS_LOG_LOGIC ("Close_link_to_" << j->m_neighborAddress);
148                 m_handleLinkFailure (j->m_neighborAddress);
149             }
150         }
151     }
152     m_nb.erase (std::remove_if (m_nb.begin (), m_nb.end (), pred), m_nb.end ());
153     m_ntimer.Cancel ();
154     m_ntimer.Schedule ();
155 }
156
157 void
158 Neighbors::ScheduleTimer ()
159 {
160     m_ntimer.Cancel ();

```

```

161     m_ntimer.Schedule ();
162 }
163
164 void
165 Neighbors::AddArpCache (Ptr<ArpCache> a)
166 {
167     m_arp.push_back (a);
168 }
169
170 void
171 Neighbors::DelArpCache (Ptr<ArpCache> a)
172 {
173     m_arp.erase (std::remove (m_arp.begin (), m_arp.end (), a), m_arp.end ());
174 }
175
176 Mac48Address
177 Neighbors::LookupMacAddress (Ipv4Address addr)
178 {
179     Mac48Address hwaddr;
180     for (std::vector<Ptr<ArpCache> >::const_iterator i = m_arp.begin ();
181          i != m_arp.end (); ++i)
182     {
183         ArpCache::Entry * entry = (*i)->Lookup (addr);
184         if (entry != 0 && (entry->IsAlive () || entry->IsPermanent ()) && !entry->IsExpired ())
185         {
186             hwaddr = Mac48Address::ConvertFrom (entry->GetMacAddress ());
187             break;
188         }
189     }
190     return hwaddr;
191 }
192
193 void
194 Neighbors::ProcessTxError (WifiMacHeader const & hdr)
195 {
196     Mac48Address addr = hdr.GetAddr1 ();
197
198     for (std::vector<Neighbor>::iterator i = m_nb.begin (); i != m_nb.end (); ++i)
199     {
200         if (i->m_hardwareAddress == addr)
201             i->close = true;
202     }
203     Purge ();
204 }
205
206 void
207 Neighbors::PrintTable (Ptr<OutputStreamWrapper> stream) const
208 {
209     int counter = 0;
210     for (std::vector<Neighbor>::const_iterator i = m_nb.begin(); i != m_nb.end(); ++i)
211     {
212         *stream->GetStream () << i->m_neighborAddress << "\t" << i->signalnoiseratio << "\t" << i->
            residualEnergy << "\t" << ++counter << "\n";
213     }
214 }
215 }
216 }

```

---

---

```

1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2  /*
3   * Copyright (c) 2009 IITP RAS
4   *
5   * This program is free software; you can redistribute it and/or modify
6   * it under the terms of the GNU General Public License version 2 as
7   * published by the Free Software Foundation;
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, write to the Free Software
16  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17  *
18  * Based on
19  * NS-2 antcomm model developed by the CMU/MONARCH group and optimized and
20  * tuned by Samir Das and Mahesh Marina, University of Cincinnati;
21  *
22  * antcomm-UU implementation by Erik Nordström of Uppsala University
23  * http://core.it.uu.se/core/index.php/antcomm-UU
24  *
25  * Authors: Elena Buchatskaia <borovkovaes@iitp.ru>
26  * Pavel Boyko <boyko@iitp.ru>
27  */
28
29 #ifndef antcommNEIGHBOR_H
30 #define antcommNEIGHBOR_H
31
32 #include "ns3/simulator.h"
33 #include "ns3/timer.h"
34 #include "ns3/ipv4-address.h"
35 #include "ns3/callback.h"
36 #include "ns3/wifi-mac-header.h"
37 #include "ns3/arp-cache.h"
38 #include <vector>
39
40 namespace ns3
41 {
42 namespace antcomm
43 {
44 class antcommProtocol;
45 /**
46  * \ingroup antcomm
47  * \brief maintain list of active neighbors
48  */
49 class Neighbors
50 {
51 public:
52     /// c-tor
53     Neighbors (Time delay);
54     /// Neighbor description
55     struct Neighbor
56     {
57         Ipv4Address m_neighborAddress;
58         Mac48Address m_hardwareAddress;
59         double signalnoiseratio;
60         double residualEnergy;
61         Time m_expireTime;
62         bool close;
63
64         Neighbor (Ipv4Address ip, Mac48Address mac, double snr, double energy, Time t) :
65             m_neighborAddress (ip), m_hardwareAddress (mac), signalnoiseratio (snr), residualEnergy (energy)
66             , m_expireTime (t),
67             close (false)
68         {
69         };
70         /// Return expire time for neighbor node with address addr, if exists, else return 0.
71         Time GetExpireTime (Ipv4Address addr);
72         /// Check that node with address addr is neighbor
73         bool IsNeighbor (Ipv4Address addr);
74         /// Update expire time for entry with address addr, if it exists, else add new entry
75         void Update (Ipv4Address addr, double snr, double energy, Time expire);
76         /// Get the degree of node
77         int Degree ();
78         /// Get the Ipv4Address of neighbor i in <vector> Neighbor, where i is an index
79         Ipv4Address GetAddress (int i);

```

```

80  /// Return the energy of neighbor node
81  double GetEnergy (Ipv4Address addr);
82  /// Return the received strength signal of addr
83  double GetRssi (Ipv4Address addr);
84  /// Remove all expired entries
85  void Purge ();
86  /// Schedule m_ntimer.
87  void ScheduleTimer ();
88  /// Remove all entries
89  void Clear () { m_nb.clear (); }
90
91  /// Add ARP cache to be used to allow layer 2 notifications processing
92  void AddArpCache (Ptr<ArpCache>);
93  /// Don't use given ARP cache any more (interface is down)
94  void DelArpCache (Ptr<ArpCache>);
95  /// Get callback to ProcessTxError
96  Callback<void, WifiMacHeader const &> GetTxErrorCallback () const { return m_txErrorCallback; }
97  /// Print the neighbor table
98  void PrintTable (Ptr<OutputStreamWrapper> stream) const;
99
100  /// Handle link failure callback
101  void SetCallback (Callback<void, Ipv4Address> cb) { m_handleLinkFailure = cb; }
102  /// Handle link failure callback
103  Callback<void, Ipv4Address> GetCallback () const { return m_handleLinkFailure; }
104
105 private:
106  /// link failure callback
107  Callback<void, Ipv4Address> m_handleLinkFailure;
108  /// TX error callback
109  Callback<void, WifiMacHeader const &> m_txErrorCallback;
110  /// Timer for neighbor's list. Schedule Purge().
111  Timer m_ntimer;
112  /// vector of entries
113  std::vector<Neighbor> m_nb;
114  /// list of ARP cached to be used for layer 2 notifications processing
115  std::vector<Ptr<ArpCache> > m_arp;
116
117  /// Find MAC address by IP using list of ARP caches
118  Mac48Address LookupMacAddress (Ipv4Address);
119  /// Process layer 2 TX error notification
120  void ProcessTxError (WifiMacHeader const &);
121 };
122
123 }
124 }
125
126 #endif /* antcommNEIGHBOR_H */

```

---

---

```

1  /*
2  * Copyright (c) 2015 SKKU Networking Laboratory
3  *
4  * Authors: Soon-gyo Jung <soongyo@skku.edu>
5  */
6  #include "antcomm-packet.h"
7  #include "ns3/address-utils.h"
8  #include "ns3/packet.h"
9  #include "ns3/ethernet-header.h"
10
11
12 namespace ns3
13 {
14
15     namespace antcomm
16     {
17     {
18         TypeHeader::TypeHeader (MessageType t):m_type (t), m_valid (true)
19         {
20         }
21
22         NS_OBJECT_ENSURE_REGISTERED (TypeHeader);
23
24         TypeId TypeHeader::GetTypeId ()
25         {
26             static TypeId tid =
27                 TypeId ("ns3::antcomm::TypeHeader").SetParent < Header >
28                 ().SetGroupName ("antcomm").AddConstructor < TypeHeader > ();
29             return tid;
30         }
31
32         TypeId TypeHeader::GetInstanceTypeId () const
33         {
34             return GetTypeId ();
35         }
36
37         uint32_t TypeHeader::GetSerializedSize () const
38         {
39             return 1;
40         }
41
42         void TypeHeader::Serialize (Buffer::Iterator i) const
43         {
44             i.WriteU8 ((uint8_t) m_type);
45         }
46
47         uint32_t TypeHeader::Deserialize (Buffer::Iterator start)
48         {
49             Buffer::Iterator i = start;
50             uint8_t type = i.ReadU8 ();
51             m_valid = true;
52             switch (type)
53             {
54                 case HELLO:
55                 case EXPL:
56                 case REPA:
57                 case RE_REPA:
58                 //case DATA_PACKET:
59                 {
60                     m_type = (MessageType) type;
61                     break;
62                 }
63                 default:
64                     m_valid = false;
65             }
66             uint32_t dist = i.GetDistanceFrom (start);
67             NS_ASSERT (dist == GetSerializedSize ());
68             return dist;
69         }
70
71         void TypeHeader::Print (std::ostream & os) const
72         {
73             switch (m_type)
74             {
75                 case HELLO:
76                 {
77                     os << "HELLO";
78                     break;
79                 }
80                 case EXPL:

```



```

81         {
82             os << "EXPL";
83             break;
84         }
85     case REPA:
86     {
87         os << "REPA";
88         break;
89     }
90     case RE_REPA:
91     {
92         os << "RE_REPA";
93         break;
94     }
95     //case DATA_PACKET:
96     //{
97     //    os << "DATA_PACKET";
98     //    break;
99     //}
100    default:
101        os << "UNKNOWN_TYPE";
102    }
103 }
104
105 bool TypeHeader::operator== (TypeHeader const &o) const
106 {
107     return (m_type == o.m_type && m_valid == o.m_valid);
108 }
109
110 std::ostream & operator<< (std::ostream & os, TypeHeader const &h)
111 {
112     h.Print (os);
113     return os;
114 }
115
116 antcommEXPLHeader::antcommEXPLHeader (uint8_t hopCount, uint8_t timeToLive, uint32_t requestID,
117     Ipv4Address origin, Ipv4Address dst):
118     m_hopCount (hopCount), m_timeToLive (timeToLive), m_requestID (requestID), m_origin (origin),
119     m_dst (dst)
120 {
121 }
122
123 NS_OBJECT_ENSURE_REGISTERED (antcommEXPLHeader);
124
125 TypeId antcommEXPLHeader::GetTypeId ()
126 {
127     static TypeId tid =
128         TypeId ("ns3::antcomm::antcommEXPLHeader").SetParent < Header >
129         ().SetGroupName ("antcomm").AddConstructor < antcommEXPLHeader > ();
130     return tid;
131 }
132
133 TypeId antcommEXPLHeader::GetInstanceTypeId () const
134 {
135     return GetTypeId ();
136 }
137
138 uint32_t antcommEXPLHeader::GetSerializedSize () const
139 {
140     return 14;
141 }
142
143 void antcommEXPLHeader::Serialize (Buffer::Iterator i) const
144 {
145     i.WriteU8 (m_hopCount);
146     i.WriteU8 (m_timeToLive);
147     i.WriteU32 (m_requestID);
148     WriteTo (i, m_origin);
149     WriteTo (i, m_dst);
150 }
151
152 uint32_t antcommEXPLHeader::Deserialize (Buffer::Iterator start)
153 {
154     Buffer::Iterator i = start;
155
156     m_hopCount = i.ReadU8 ();
157     m_timeToLive = i.ReadU8 ();
158     m_requestID = i.ReadU32 ();
159     ReadFrom (i, m_origin);
160     ReadFrom (i, m_dst);

```

```

159
160     uint32_t dist = i.GetDistanceFrom (start);
161     NS_ASSERT (dist == GetSerializedSize ());
162     return dist;
163 }
164
165 void antcommEXPLHeader::Print (std::ostream & os) const
166 {
167     os << "antcomm-";
168     os << "HopCount:" << m_hopCount << "TTL:" << m_timeToLive << "AntID" << m_requestID << "
        SourceID:" << m_origin << "Destination:" << m_dst;
169 }
170
171 bool antcommEXPLHeader::operator== (antcommEXPLHeader const &o) const
172 {
173     return (m_hopCount == o.m_hopCount && m_timeToLive == o.m_timeToLive && m_requestID == o.
        m_requestID
        && m_origin == o.m_origin && m_dst == o.m_dst );
174 }
175
176 std::ostream & operator<< (std::ostream & os, antcommEXPLHeader const &h)
177 {
178     h.Print (os);
179     return os;
180 }
181
182
183
184 antcommHelloHeader::antcommHelloHeader (uint8_t timeToLive, uint8_t hopCount, double
    residualEnergy, Ipv4Address origin, Ipv4Address dst, Time lifeTime):
185     m_timeToLive (timeToLive), m_hopCount (hopCount), m_residualEnergy (residualEnergy),
186     m_origin (origin), m_dst (dst)
187 {
188     m_lifeTime = uint32_t (lifeTime.GetMilliSeconds ());
189 }
190
191 NS_OBJECT_ENSURE_REGISTERED (antcommHelloHeader);
192
193 TypeId antcommHelloHeader::GetTypeId ()
194 {
195     static TypeId tid =
196         TypeId ("ns3::antcomm::antcommHelloHeader").SetParent < Header >
197         ().SetGroupName ("antcomm").AddConstructor < antcommHelloHeader > ();
198     return tid;
199 }
200
201 TypeId antcommHelloHeader::GetInstanceTypeId () const
202 {
203     return GetTypeId ();
204 }
205
206 uint32_t antcommHelloHeader::GetSerializedSize () const
207 {
208     return 18 /*10 */ ;
209 }
210
211 void antcommHelloHeader::Serialize (Buffer::Iterator i) const
212 {
213
214     i.WriteU8 (m_timeToLive);
215     i.WriteU8 (m_hopCount);
216     //i.WriteU32 (m_pheromone*100000000.0);
217     i.WriteU32 (m_residualEnergy*100000000.0);
218     WriteTo (i, m_origin);
219     WriteTo (i, m_dst);
220     i.WriteHtonU32 (m_lifeTime);
221 }
222
223
224 uint32_t antcommHelloHeader::Deserialize (Buffer::Iterator start)
225 {
226     Buffer::Iterator i = start;
227
228     m_timeToLive = i.ReadU8 ();
229     m_hopCount = i.ReadU8 ();
230     // m_pheromone = i.ReadU32 ()/100000000.0; //
231     m_residualEnergy = i.ReadU32 ()/100000000.0; //
232     ReadFrom (i, m_origin);
233     ReadFrom (i, m_dst);
234     m_lifeTime = i.ReadNtohU32 ();
235

```

```

236
237     uint32_t dist = i.GetDistanceFrom (start);
238     NS_ASSERT (dist == GetSerializedSize ());
239     return dist;
240 }
241
242 void antcommHelloHeader::Print (std::ostream & os) const
243 {
244     os << "TTL:" << (int) m_timeToLive << ", Hops:" << m_hopCount
245         << ", ResidualEnergy" << m_residualEnergy << ", Src:" << m_origin << ", Dst:" << m_dst << ", LifeTime" << m_lifeTime;
246     /* " Residual Energy : " << m_residualEnergy << */
247 }
248
249 void antcommHelloHeader::SetLifeTime (Time t)
250 {
251     m_lifeTime = t.GetMilliSeconds ();
252 }
253
254 Time antcommHelloHeader::GetLifeTime () const
255 {
256     Time t (MilliSeconds (m_lifeTime));
257     return t;
258 }
259
260
261 bool antcommHelloHeader::operator== (antcommHelloHeader const &o) const
262 {
263     return (m_timeToLive == o.m_timeToLive && m_hopCount == o.m_hopCount
264         && m_residualEnergy == o.m_residualEnergy && m_origin == o.m_origin && m_dst == o.m_dst
265         && m_lifeTime == o.m_lifeTime);
266 }
267
268 std::ostream & operator<< (std::ostream & os, antcommHelloHeader const &h)
269 {
270     h.Print (os);
271     return os;
272 }
273
274 antcommREPAHeader::antcommREPAHeader (uint8_t hopCount, uint8_t timeToLive, Ipv4Address origin,
275     Ipv4Address dst) : //
276 {
277     m_hopCount (hopCount), m_timeToLive (timeToLive), m_origin (origin), m_dst (dst)
278 }
279
280 NS_OBJECT_ENSURE_REGISTERED (antcommREPAHeader);
281
282 TypeId antcommREPAHeader::GetTypeId ()
283 {
284     static TypeId tid =
285         TypeId ("ns3::antcomm::antcommREPAHeader").SetParent < Header >
286         ().SetGroupName ("antcomm").AddConstructor < antcommREPAHeader > ();
287     return tid;
288 }
289
290 TypeId antcommREPAHeader::GetInstanceTypeId () const
291 {
292     return GetTypeId ();
293 }
294
295 uint32_t antcommREPAHeader::GetSerializedSize () const
296 {
297     return 10 ;
298 }
299
300 void antcommREPAHeader::Serialize (Buffer::Iterator i) const
301 {
302     i.WriteU8 (m_hopCount);
303     i.WriteU8 (m_timeToLive);
304     //i.WriteU16 (m_sourceID);
305     WriteTo (i, m_origin);
306     WriteTo (i, m_dst);
307 }
308
309 uint32_t antcommREPAHeader::Deserialize (Buffer::Iterator start)
310 {
311     Buffer::Iterator i = start;
312     m_hopCount = i.ReadU8 ();
313     m_timeToLive = i.ReadU8 ();

```

```

313     //m_sourceID = i.ReadU16 ();
314     ReadFrom (i, m_origin);
315     ReadFrom (i, m_dst);
316
317     uint32_t dist = i.GetDistanceFrom (start);
318     NS_ASSERT (dist == GetSerializedSize ());
319     return dist;
320 }
321
322 void antcommREPAHeader::Print (std::ostream & os) const
323 {
324     os << "antcomm-";
325     os << "HopCount:" << m_hopCount << "TTL" << m_timeToLive << "SourceID:" << m_origin << "
        Destination" << m_dst ;
326 }
327
328 bool antcommREPAHeader::operator== (antcommREPAHeader const &o) const
329 {
330     return (m_hopCount == o.m_hopCount && m_timeToLive == o.m_timeToLive && m_origin == o.m_origin
        && m_dst == o.m_dst);
331 }
332
333 std::ostream & operator<< (std::ostream & os, antcommREPAHeader const &h)
334 {
335     h.Print (os);
336     return os;
337 }
338
339
340
341 antcommREREPAHeader::antcommREREPAHeader (uint8_t hopCount, uint8_t timeToLive, double pheromone,
        double residualEnergy, Ipv4Address origin, Ipv4Address dst):
342     m_hopCount (hopCount), m_timeToLive (timeToLive), m_pheromone (pheromone),
343     m_residualEnergy (residualEnergy),
344     m_origin (origin), m_dst (dst)
345 {
346 }
347
348 NS_OBJECT_ENSURE_REGISTERED (antcommREREPAHeader);
349
350 TypeId antcommREREPAHeader::GetTypeId ()
351 {
352     static TypeId tid =
353         TypeId ("ns3::antcomm::antcommREREPAHeader").SetParent < Header >
354         ().SetGroupName ("antcomm").AddConstructor < antcommREREPAHeader > ();
355     return tid;
356 }
357
358 TypeId antcommREREPAHeader::GetInstanceTypeId () const
359 {
360     return GetTypeId ();
361 }
362
363 uint32_t antcommREREPAHeader::GetSerializedSize () const
364 {
365     return 18 /*32 */ ;
366 }
367
368 void antcommREREPAHeader::Serialize (Buffer::Iterator i) const
369 {
370     i.WriteU8 (m_hopCount);
371     i.WriteU8 (m_timeToLive);
372     i.WriteU32 (m_pheromone * 100000000);
373     i.WriteU32 (m_residualEnergy * 100000000);
374     //i.WriteU16 (m_sourceID);
375     WriteTo (i, m_origin);
376     WriteTo (i, m_dst);
377 }
378
379 uint32_t antcommREREPAHeader::Deserialize (Buffer::Iterator start)
380 {
381     Buffer::Iterator i = start;
382
383     m_hopCount = i.ReadU8 ();
384     m_timeToLive = i.ReadU8 ();
385     m_pheromone = i.ReadU32 () / 100000000.0;
386     m_residualEnergy = i.ReadU32 () / 100000000.0;
387     //m_sourceID = i.ReadU16 ();
388     ReadFrom (i, m_origin);
389     ReadFrom (i, m_dst);

```

```

390
391
392     uint32_t dist = i.GetDistanceFrom (start);
393     NS_ASSERT (dist == GetSerializedSize ());
394     return dist;
395 }
396
397 void antcommREREPAHeader::Print (std::ostream & os) const
398 {
399     os << "antcomm_";
400     os << "HopCount:" << m_hopCount << "TTL" << m_timeToLive << "Pheromone:" << m_pheromone
401         <<
402         "ResidualEnergy:" << m_residualEnergy << "SourceID:" <<
403         m_origin << "Destination" << m_dst ;
404 }
405
406 bool antcommREREPAHeader::operator== (antcommREREPAHeader const &o) const
407 {
408     return (m_hopCount == o.m_hopCount && m_pheromone == o.m_pheromone
409         && m_residualEnergy == o.m_residualEnergy
410         && m_residualEnergy == o.m_residualEnergy
411         && m_origin == o.m_origin && m_dst == o.m_dst && m_timeToLive == o.m_timeToLive );
412 }
413
414 std::ostream & operator<< (std::ostream & os,
415     antcommREREPAHeader const &h)
416 {
417     h.Print (os);
418     return os;
419 }
420
421 antcommDataPacketHeader::antcommDataPacketHeader (Ipv4Address origin, Ipv4Address dst, uint8_t
422     hopCount, uint8_t timeToLive, uint16_t hopDistance, uint32_t packetID): //
423     m_origin (origin ), m_dst (dst ), m_hopCount (hopCount), m_timeToLive (timeToLive),
424     m_hopDistance (hopDistance), m_packetID (packetID)
425 {
426 }
427
428 NS_OBJECT_ENSURE_REGISTERED (antcommDataPacketHeader);
429
430 TypeId antcommDataPacketHeader::GetTypeId ()
431 {
432     static TypeId tid =
433         TypeId ("ns3::antcomm::antcommDataPacketHeader").SetParent < Header >
434         ().SetGroupName ("antcomm").AddConstructor < antcommDataPacketHeader >
435         ();
436     return tid;
437 }
438
439 TypeId antcommDataPacketHeader::GetInstanceTypeId () const
440 {
441     return GetTypeId ();
442 }
443
444 uint32_t antcommDataPacketHeader::GetSerializedSize () const
445 {
446     return 512 ;
447 }
448
449 void antcommDataPacketHeader::Serialize (Buffer::Iterator i) const
450 {
451     //i.WriteU16 (m_sourceID);
452     //i.WriteU16 (m_destinationID);
453     WriteTo (i, m_origin);
454     WriteTo (i, m_dst);
455     i.WriteU8 (m_hopCount);
456     i.WriteU8 (m_timeToLive);
457     i.WriteU16 (m_hopDistance);
458     i.WriteU32 (m_packetID);
459 }
460
461 uint32_t antcommDataPacketHeader::Deserialize (Buffer::Iterator start)
462 {
463     Buffer::Iterator i = start;
464
465     //m_sourceID = i.ReadU16 ();
466     //m_destinationID = i.ReadU16 ();
467     ReadFrom (i, m_origin);
468     ReadFrom (i, m_dst);

```

```

467     m_hopCount = i.ReadU8 ();
468     m_timeToLive = i.ReadU8 ();
469     m_hopDistance = i.ReadU16 ();
470     m_packetID = i.ReadU32();
471
472
473     uint32_t dist = i.GetDistanceFrom (start);
474     return dist;
475 }
476
477 void antcommDataPacketHeader::Print (std::ostream & os) const
478 {
479     os << "antcomm-";
480     os << "SourceID:" << m_origin << "DestinationID:" << m_dst << "Hopcount" << m_hopCount
        << "TTL" << m_timeToLive << "HopDistance" << m_hopDistance << "PacketID" << m_packetID ;
481 }
482
483 bool antcommDataPacketHeader::operator== (antcommDataPacketHeader const &o) const
484 {
485     return (m_origin == o.m_origin && m_dst == o.m_dst && m_hopCount == o.m_hopCount && m_timeToLive
        == o.m_timeToLive && m_hopDistance == o.m_hopDistance && m_packetID == o.m_packetID );
486 }
487
488 std::ostream & operator<< (std::ostream & os,
489                             antcommDataPacketHeader const &h)
490 {
491     h.Print (os);
492     return os;
493 }
494 }
495 }

```

---

```

1  /*
2   * Copyright (c) 2015 SKKU Networking Laboratory
3   *
4   * Authors: Soon-gyo Jung <soongyo@skku.edu>
5   */
6 #ifndef antcommPACKET_H
7 #define antcommPACKET_H
8
9 #include <iostream>
10 #include "ns3/header.h"
11 #include "ns3/enum.h"
12 #include "ns3/ipv4-address.h"
13 #include <map>
14 #include "ns3/nstime.h"
15 #include "ns3/ethernet-header.h"
16
17 namespace ns3
18 {
19
20     namespace antcomm
21     {
22
23         enum MessageType
24         {
25             HELLO = 1,    //!< EXPL
26             EXPL = 2,     //!< REPA
27             REPA = 3,     //!< RE_REPA
28             RE_REPA = 4,
29             //DATA_PACKET = 5
30         };
31
32         /**
33          * \ingroup antcomm
34          * \brief antcomm types
35          */
36         class TypeHeader : public Header
37         {
38         public:
39             /// c-tor
40             TypeHeader (MessageType t = EXPL);
41
42             // Header serialization/deserialization
43             static TypeId GetTypeId ();
44             TypeId GetInstanceTypeId () const;
45             uint32_t GetSerializedSize () const;
46             void Serialize (Buffer::Iterator start) const;
47             uint32_t Deserialize (Buffer::Iterator start);
48             void Print (std::ostream & os) const;
49
50             /// Return type
51             MessageType Get () const
52             {
53                 return m_type;
54             }
55             /// Check that type if valid
56             bool IsValid () const
57             {
58                 return m_valid;
59             }
60             bool operator== (TypeHeader const &o) const;
61 private:
62             MessageType m_type;
63             bool m_valid;
64
65             std::ostream & operator<< (std::ostream & os, TypeHeader const &h);
66
67             /**
68              * \ingroup antcomm
69              * \brief antcomm EXPL Message Format
70              * \verbatim
71              0 1 2 3
72              0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
73              +-----+-----+-----+-----+-----+-----+-----+-----+
74              | Type | Hop Count | Time to live | Void |
75              +-----+-----+-----+-----+-----+-----+-----+-----+
76              | RREQ ID |
77              +-----+-----+-----+-----+-----+-----+-----+-----+
78              | Destination IP address |
79              +-----+-----+-----+-----+-----+-----+-----+-----+

```

```

81  / Originator IP address /
82  +-----+
83
84
85  \endverbatim
86  */
87
88  class antcommEXPLHeader:public Header
89  {
90  public:
91      /// c-tor
92      antcommEXPLHeader (uint8_t hopCount = 0, uint8_t timeToLive = 0, uint32_t requestID = 0,
93                          Ipv4Address origin = Ipv4Address (), Ipv4Address dst = Ipv4Address ());
94
95      // Header serialization/deserialization
96      static TypeId GetTypeId ();
97      TypeId GetInstanceTypeId () const;
98      uint32_t GetSerializedSize () const;
99      void Serialize (Buffer::Iterator start) const;
100     uint32_t Deserialize (Buffer::Iterator start);
101     void Print (std::ostream & os) const;
102
103     // Fields
104     void SetHopCount (uint8_t count)
105     {
106         m_hopCount = count;
107     }
108     uint8_t GetHopCount () const
109     {
110         return m_hopCount;
111     }
112     void SetId (uint32_t id)
113     {
114         m_requestID = id;
115     }
116     uint32_t GetId () const
117     {
118         return m_requestID;
119     }
120     void SetOrigin (Ipv4Address a)
121     {
122         m_origin = a;
123     }
124     Ipv4Address GetOrigin () const
125     {
126         return m_origin;
127     }
128     void SetDst (Ipv4Address a)
129     {
130         m_dst = a;
131     }
132     Ipv4Address GetDst () const
133     {
134         return m_dst;
135     }
136     void SetTimeToLive (uint8_t timeToLive)
137     {
138         m_timeToLive = timeToLive;
139     }
140     uint8_t GetTimeToLive () const
141     {
142         return m_timeToLive;
143     }
144
145     //void SetSourceID (uint16_t sourceID) { m_sourceID = sourceID; } //
146     //uint16_t GetSourceID () const { return m_sourceID; } //
147
148     bool operator== (antcommEXPLHeader const &o) const;
149
150 private:
151     uint8_t m_hopCount; ///< Hop Count
152     uint8_t m_timeToLive;
153     uint32_t m_requestID;
154     Ipv4Address m_origin;
155     Ipv4Address m_dst;
156
157
158
159 };
160

```



```

161     std::ostream & operator<< (std::ostream & os, antcommEXPLHeader const &h);
162
163
164
165     /**
166     * \ingroup antcomm
167     * \brief antcomm Hello Message Format
168     * \verbatim
169     0 1 2 3
170     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
171     +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
172     | Type | Time to Live | Hop Count | Void |
173     +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
174     | Destination IP address |
175     +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
176     | Originator IP address |
177     +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
178     | Residual Energy (32 bits) |
179     +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
180     | Lifetime (32 bits) |
181     +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
182
183     \endverbatim
184     */
185     class antcommHelloHeader:public Header
186     {
187     public:
188         /// c-tor
189         antcommHelloHeader (uint8_t timeToLive = 0, uint8_t hopCount = 0, double residualEnergy = 0,
190                             Ipv4Address origin = Ipv4Address (), Ipv4Address dst = Ipv4Address (), Time lifetime =
191                             MilliSeconds (0));
192
193         // Header serialization/deserialization
194         static TypeId GetTypeId ();
195         TypeId GetInstanceTypeId () const;
196         uint32_t GetSerializedSize () const;
197         void Serialize (Buffer::Iterator start) const;
198         uint32_t Deserialize (Buffer::Iterator start);
199         void Print (std::ostream & os) const;
200
201         // Fields
202
203         void SetTimeToLive (uint8_t timeToLive)
204         {
205             m_timeToLive = timeToLive;
206         }
207         uint8_t GetTimeToLive () const
208         {
209             return m_timeToLive;
210         }
211         void SetHopCount (uint8_t hopCount)
212         {
213             m_hopCount = hopCount;
214         }
215         uint8_t GetHopCount () const
216         {
217             return m_hopCount;
218         }
219         void SetResidualEnergy ( double residualEnergy)
220         {
221             m_residualEnergy = residualEnergy;
222         }
223         double GetResidualEnergy() const
224         {
225             return m_residualEnergy;
226         }
227         void SetOrigin (Ipv4Address a)
228         {
229             m_origin = a;
230         }
231         Ipv4Address GetOrigin () const
232         {
233             return m_origin;
234         }
235         void SetDst (Ipv4Address a)
236         {
237             m_dst = a;
238         }
239         Ipv4Address GetDst () const

```

```

240     {
241     return m_dst;
242     }
243     void SetLifeTime (Time t);
244     Time GetLifeTime () const;
245
246
247     bool operator== (antcommHelloHeader const &o) const;
248
249     private:
250
251
252     uint8_t m_timeToLive;
253     uint8_t m_hopCount;
254     double m_residualEnergy;
255     Ipv4Address m_origin;
256     Ipv4Address m_dst;
257     uint32_t m_lifeTime;
258     };
259
260     std::ostream & operator<< (std::ostream & os,
261                               antcommHelloHeader const &h);
262
263
264     /**
265     * \ingroup antcomm
266     * \brief antcomm REPA Message Format
267     \verbatim
268     0 1 2 3
269     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
270     +-----+-----+-----+-----+-----+-----+-----+-----+
271     | Type | Hop Count | Time to live | Void |
272     +-----+-----+-----+-----+-----+-----+-----+-----+
273     | Destination IP address |
274     +-----+-----+-----+-----+-----+-----+-----+-----+
275     | Originator IP address |
276     +-----+-----+-----+-----+-----+-----+-----+-----+
277
278     \endverbatim
279     */
280
281     class antcommREPAHeader:public Header
282     {
283     public:
284         /// c-tor
285         antcommREPAHeader (uint8_t hopCount = 0, uint8_t timeToLive = 0, Ipv4Address origin =
286                             Ipv4Address (), Ipv4Address dst = Ipv4Address () );
287
288         // Header serialization/deserialization
289         static TypeId GetTypeId ();
290         TypeId GetInstanceTypeId () const;
291         uint32_t GetSerializedSize () const;
292         void Serialize (Buffer::Iterator start) const;
293         uint32_t Deserialize (Buffer::Iterator start);
294         void Print (std::ostream & os) const;
295
296         void SetHopCount (uint8_t hopCount)
297         {
298             m_hopCount = hopCount;
299         }
300         uint16_t GetHopCount () const
301         {
302             return m_hopCount;
303         }
304         void SetOrigin (Ipv4Address a)
305         {
306             m_origin = a;
307         }
308         Ipv4Address GetOrigin () const
309         {
310             return m_origin;
311         }
312         void SetDst (Ipv4Address a)
313         {
314             m_dst = a;
315         }
316         Ipv4Address GetDst () const
317         {
318             return m_dst;
319         }
320         void SetTimeToLive (uint8_t timeToLive)

```

```

320     {
321         m_timeToLive = timeToLive;
322     }
323     uint8_t GetTimeToLive () const
324     {
325         return m_timeToLive;
326     }
327     //void SetSourceID (uint16_t sourceID) { m_sourceID = sourceID; } //
328     //uint16_t GetSourceID () const { return m_sourceID; } //
329
330     bool operator== (antcommREPAHeader const &o) const;
331
332 private:
333     uint8_t m_hopCount; ///< Hop Count
334     uint8_t m_timeToLive;
335     Ipv4Address m_origin;
336     Ipv4Address m_dst;
337
338 };
339
340 std::ostream & operator<< (std::ostream & os, antcommREPAHeader const &h);
341
342
343 /**
344 * \ingroup antcomm
345 * \brief antcomm REREPA Message Format
346 * \verbatim
347 0 1 2 3
348 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
349 +-----+
350 | Type | Hop Count | Time to live | Void |
351 +-----+
352 | Destination IP address |
353 +-----+
354 | Originator IP address |
355 +-----+
356 | Payload |
357 +-----+
358
359 \endverbatim
360 */
361 class antcommREREPAHeader:public Header
362 {
363 public:
364     /// c-tor
365     antcommREREPAHeader (uint8_t hopCount = 0, uint8_t timeToLive = 0, double pheromone = 0.0,
366         double residualEnergy = 0.0, Ipv4Address origin = Ipv4Address (), Ipv4Address dst =
367         Ipv4Address ());
368
369     // Header serialization/deserialization
370     static TypeId GetTypeId ();
371     TypeId GetInstanceTypeId () const;
372     uint32_t GetSerializedSize () const;
373     void Serialize (Buffer::Iterator start) const;
374     uint32_t Deserialize (Buffer::Iterator start);
375     void Print (std::ostream & os) const;
376
377     void SetHopCount (uint8_t hopCount)
378     {
379         m_hopCount = hopCount;
380     }
381     uint16_t GetHopCount () const
382     {
383         return m_hopCount;
384     }
385     void SetPheromone (double pheromone)
386     {
387         m_pheromone = pheromone;
388     }
389     double GetPheromone () const
390     {
391         return m_pheromone;
392     }
393     void SetResidualEnergy (double residualEnergy)
394     {
395         m_residualEnergy = residualEnergy;
396     }
397     double GetResidualEnergy () const
398     {
399         return m_residualEnergy;
400     }

```

```

399     }
400     void SetOrigin (Ipv4Address a)
401     {
402         m_origin = a;
403     }
404     Ipv4Address GetOrigin () const
405     {
406         return m_origin;
407     }
408     void SetDst (Ipv4Address a)
409     {
410         m_dst = a;
411     }
412     Ipv4Address GetDst () const
413     {
414         return m_dst;
415     }
416     void SetTimeToLive (uint8_t timeToLive)
417     {
418         m_timeToLive = timeToLive;
419     }
420     uint8_t GetTimeToLive () const
421     {
422         return m_timeToLive;
423     }
424     //void SetSourceID (uint16_t sourceID) { m_sourceID = sourceID; } //
425     //uint16_t GetSourceID () const { return m_sourceID; } //
426
427     bool operator== (antcommREREPAHeader const &o) const;
428
429 private:
430     uint8_t m_hopCount; ///< Hop Count
431     uint8_t m_timeToLive;
432     double m_pheromone;
433     double m_residualEnergy;
434     Ipv4Address m_origin;
435     Ipv4Address m_dst;
436
437
438
439
440
441
442 };
443
444 std::ostream & operator<< (std::ostream & os,
445                             antcommREREPAHeader const &h);
446
447
448 /**
449 * \ingroup antcomm
450 * \brief antcomm DATAPacket Message Format
451 * \verbatim
452 0 1 2 3
453 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
454 +-----+-----+-----+-----+-----+-----+-----+-----+
455 | Type | Hop Count | Time to live | Void |
456 +-----+-----+-----+-----+-----+-----+-----+-----+
457 | Destination IP address |
458 +-----+-----+-----+-----+-----+-----+-----+-----+
459 | Originator IP address |
460 +-----+-----+-----+-----+-----+-----+-----+-----+
461 | PacketID |
462 +-----+-----+-----+-----+-----+-----+-----+-----+
463 | Hop Distance |
464 +-----+-----+-----+-----+-----+-----+
465
466 \endverbatim
467 */
468 class antcommDataPacketHeader:public Header
469 {
470 public:
471     /// c-tor
472     antcommDataPacketHeader (Ipv4Address origin = Ipv4Address (), Ipv4Address dst = Ipv4Address (),
473                             uint8_t hopCount = 0, uint8_t timeToLive = 0, uint16_t hopDistance = 0, uint32_t packetID =
474                             0); // 0
475
476     // Header serialization/deserialization
477     static TypeId GetTypeId ();
478     TypeId GetInstanceTypeId () const;
479     uint32_t GetSerializedSize () const;

```

```

478 void Serialize (Buffer::Iterator start) const;
479 uint32_t Deserialize (Buffer::Iterator start);
480 void Print (std::ostream & os) const;
481
482 // Fields
483 //void SetSourceID (uint16_t sourceID) { m_sourceID = sourceID; } //
484 //uint16_t GetSourceID () const { return m_sourceID; } //
485 //void SetDestinationID (uint16_t destinationID) { m_destinationID = destinationID; } //
486 //uint16_t GetDestinationID () const { return m_destinationID; } //
487
488 void SetOrigin (Ipv4Address a)
489 {
490     m_origin = a;
491 }
492 Ipv4Address GetOrigin () const
493 {
494     return m_origin;
495 }
496
497 void SetDst (Ipv4Address a)
498 {
499     m_dst = a;
500 }
501 Ipv4Address GetDst () const
502 {
503     return m_dst;
504 }
505 void SetTimeToLive (uint8_t timeToLive)
506 {
507     m_timeToLive = timeToLive;
508 }
509 uint8_t GetTimeToLive () const
510 {
511     return m_timeToLive;
512 }
513 void SetHopCount (uint8_t hopCount)
514 {
515     m_hopCount = hopCount;
516 }
517 uint16_t GetHopCount () const
518 {
519     return m_hopCount;
520 }
521 void SetHopDistance (uint16_t hopDistance) { m_hopDistance = hopDistance; }
522 uint16_t GetHopDistance () const { return m_hopDistance; }
523 void SetPacketID (uint32_t packetID) { m_packetID = packetID; }
524 uint32_t GetPacketID () const { return m_packetID; }
525
526 bool operator== (antcommDataPacketHeader const &o) const;
527
528 private:
529     //uint16_t m_sourceID; ///< Source ID //
530     //uint16_t m_destinationID; ///< Destination ID //
531     Ipv4Address m_origin;
532     Ipv4Address m_dst;
533     uint8_t m_hopCount;
534     uint8_t m_timeToLive;
535     uint16_t m_hopDistance;
536     uint32_t m_packetID;
537
538 };
539
540 std::ostream & operator<< (std::ostream & os,
541                             antcommDataPacketHeader const &h);
542
543 }
544 }
545
546 #endif /* antcommPACKET_H */

```

---

---

```

1  /*
2  * Copyright (c) 2015 SKKU Networking Laboratory
3  *
4  * Authors: Soon-gyo Jung <soongyo@skku.edu>
5  */
6  #define NS_LOG_APPEND_CONTEXT      \
7      if (m_ipv4) { std::clog << "[node_" << m_ipv4->GetObject<Node> ()->GetId () << "]"<< "; }
8
9  #include "antcomm-helper.h"
10 #include "antcomm-protocol.h"
11 #include "ns3/core-module.h"
12 #include "ns3/log.h"
13 #include "ns3/boolean.h"
14 #include "ns3/random-variable-stream.h"
15 #include "ns3/inet-socket-address.h"
16 #include "ns3/trace-source-accessor.h"
17 #include "ns3/udp-socket-factory.h"
18 #include "ns3/wifi-net-device.h"
19 #include "ns3/adhoc-wifi-mac.h"
20 #include "ns3/string.h"
21 #include "ns3/pointer.h"
22 #include "ns3/energy-module.h"
23 #include "ns3/propagation-loss-model.h"
24 #include "ns3/snr-tag.h"
25 #include "ns3/constant-position-mobility-model.h"
26 #include "ns3/random-variable-stream.h"
27 #include "ns3/ipv4-interface.h"
28 #include <algorithm>
29 #include <limits>
30 #include <math.h>
31 #include <iostream>
32 #include <stdio.h>
33 #include <iomanip>
34 #define LAMBDA 0.7
35
36 namespace ns3
37 {
38
39     NS_LOG_COMPONENT_DEFINE ("antcommProtocol");
40
41     namespace antcomm
42     {
43         NS_OBJECT_ENSURE_REGISTERED (antcommProtocol);
44
45         /// UDP Port for antcomm control traffic
46         const uint32_t antcommProtocol::antcomm_PORT = 285;
47
48         //-----
49         /// Tag used by antcomm implementation
50
51         class DeferredRouteOutputTag:public Tag
52         {
53
54         public:
55             DeferredRouteOutputTag (int32_t o = -1):Tag (), m_oif (o) {}
56
57             static TypeId GetTypeId ()
58             {
59                 static TypeId tid = TypeId ("ns3::antcomm::DeferredRouteOutputTag")
60                     .SetParent < Tag > ()
61                     .SetGroupName ("antcomm")
62                     .AddConstructor <DeferredRouteOutputTag > ()
63                     ;
64                 return tid;
65             }
66
67             TypeId GetInstanceTypeId () const
68             {
69                 return GetTypeId ();
70             }
71
72             int32_t GetInterface () const
73             {
74                 return m_oif;
75             }
76
77             void SetInterface (int32_t oif)
78             {
79                 m_oif = oif;
80             }

```

```

81
82     uint32_t GetSerializedSize () const
83     {
84         return sizeof (int32_t);
85     }
86
87     void Serialize (TagBuffer i) const
88     {
89         i.WriteU32 (m_oif);
90     }
91
92     void Deserialize (TagBuffer i)
93     {
94         m_oif = i.ReadU32 ();
95     }
96
97     void Print (std::ostream & os) const
98     {
99         os << "DeferredRouteOutputTag: output interface=" << m_oif;
100     }
101
102 private:
103     /// Positive if output device is fixed in RouteOutput
104     int32_t m_oif;
105 };
106
107 NS_OBJECT_ENSURE_REGISTERED (DeferredRouteOutputTag);
108
109 ///-----
110
111 antcommProtocol::antcommProtocol ():
112     m_numberOfReceivedPacket (0),
113     EXPLRateLimit (10),
114     m_averageHopDistanceOfReceivedPackets (0.0),
115     m_averageEndToEndDelays (0.0),
116     m_numberOfReceivedAnts (0.0),
117     ActiveRouteTimeout (Seconds (3)),
118     NetDiameter (35),
119     NodeTraversalTime (MilliSeconds (40)),
120     NetTraversalTime (Time ((2 * NetDiameter) * NodeTraversalTime)),
121     PathDiscoveryTime (Time (2 * NetTraversalTime)),
122     m_currentPhase (EXPLORATION),
123     HelloInterval (Seconds (0.5)),
124     AllowedHelloLoss (2),
125     m_explorationTime (EXPLORATION_TIME),
126     m_explorationTimer (Timer::CANCEL_ON_DESTROY),
127     m_repairTime (REPAIR_TIME),
128     m_repairTimer (Timer::CANCEL_ON_DESTROY),
129     m_transmissionRange (TRANSMISSION_RANGE),
130     m_distanceForTimer (DISTANCE_FOR_TIMER),
131     m_pheromone (INITIAL_PHEROMONE),
132     m_pheromoneDecayRate (PHEROMONE_DECAY_RATE),
133     m_updatedTimeOfPheromone (Seconds (0)),
134     m_epsilon (EPSILON),
135     m_timeMinForWaitingTimer (TIME_MIN_FOR_WAITING_TIMER),
136     m_timeMaxForWaitingTimer (TIME_MAX_FOR_WAITING_TIMER), m_alpha (ALPHA),
137     m_beta (BETA), m_gamma (GAMMA), m_routingTable (), MaxQueueLen (64),
138     MaxQueueTime (Seconds (30)), m_queue (MaxQueueLen, MaxQueueTime),
139     //m_helloInterval (TIME_FOR_HELLO_TIMER),
140     m_requestId (0),
141     m_helloTimer (Timer::CANCEL_ON_DESTROY),
142     m_rrreqIdCache (PathDiscoveryTime),
143     m_dpd (PathDiscoveryTime),
144     m_explCount (0),
145     m_neighborTable (HelloInterval),
146     m_lastBcastTime (Seconds (0)),
147     m_forwardTimer (Timer::CANCEL_ON_DESTROY),
148     m_backwardTimer (Timer::CANCEL_ON_DESTROY),
149     m_retryTimer (Timer::CANCEL_ON_DESTROY),
150     numberOfDeferredPacket (0), m_waitingTimer (Timer::CANCEL_ON_DESTROY)
151 {
152 }
153
154
155 TypeId antcommProtocol::GetTypeId (void)
156 {
157     static TypeId tid = TypeId ("ns3::antcomm::antcommProtocol")
158     .SetParent <Ipv4RoutingProtocol > ()
159     .SetGroupName ("antcomm")
160     .AddConstructor <antcommProtocol > ()

```

```

161 .AddAttribute ("ExplorationTime", "The_exploration_time.",
162             UIntegerValue (EXPLORATION_TIME),
163             MakeUIntegerAccessor(&antcommProtocol::m_explorationTime),
164             MakeUIntegerChecker < uint32_t >())
165 .AddAttribute ("HelloInterval", "HELLO_messages_emission_interval.",
166             TimeValue (Seconds (5)),
167             MakeTimeAccessor (&antcommProtocol::HelloInterval),
168             MakeTimeChecker ())
169 .AddAttribute ("NetDiameter", "Net_diameter_measures_the_maximum_possible_number_of_hops_between
            two_nodes_in_the_network",
170             UIntegerValue (35),
171             MakeUIntegerAccessor(&antcommProtocol::NetDiameter),
172             MakeUIntegerChecker <uint32_t>())
173 .AddAttribute ("NodeTraversalTime", "Conservative_estimate_of_the_average_one_hop_traversal_time
            for_packets_and_should_include"
174             "queuing_delays_interrupt_processing_times_and_transfer_times.",
175             TimeValue (MilliSeconds (40)),
176             MakeTimeAccessor(&antcommProtocol::NodeTraversalTime),
177             MakeTimeChecker ())
178 .AddAttribute ("NetTraversalTime", "Estimate_of_the_average_net_traversal_time=2*_
            NodeTraversalTime*_NetDiameter",
179             TimeValue (Seconds (2.8)),
180             MakeTimeAccessor (&antcommProtocol::NetTraversalTime),
181             MakeTimeChecker ())
182 .AddAttribute ("ActiveRouteTimeout", "Period_of_time_during_which_the_route_is_considered_to_be
            valid",
183             TimeValue (Seconds (3)),
184             MakeTimeAccessor (&antcommProtocol::ActiveRouteTimeout),
185             MakeTimeChecker ())
186 .AddAttribute ("DistanceForTimer", "The_distance_for_timer.",
187             DoubleValue (0.5),
188             MakeDoubleAccessor(&antcommProtocol::m_distanceForTimer),
189             MakeDoubleChecker <double >())
190 .AddAttribute ("PathDiscoveryTime", "Estimate_of_maximum_time_needed_to_find_route_in_network=
            2*_NetTraversalTime",
191             TimeValue (Seconds (5.6)),
192             MakeTimeAccessor (&antcommProtocol::PathDiscoveryTime),
193             MakeTimeChecker ())
194 .AddAttribute ("MaxQueueLen", "Maximum_number_of_packets_that_we_allow_a_routing_protocol_to
            buffer.",
195             UIntegerValue (64),
196             MakeUIntegerAccessor (&antcommProtocol::SetMaxQueueLen,
197                                 &antcommProtocol::GetMaxQueueLen),
198             MakeUIntegerChecker <uint32_t>())
199 .AddAttribute ("MaxQueueTime", "Maximum_time_packets_can_be_queued_(in_seconds)",
200             TimeValue (Seconds (30)),
201             MakeTimeAccessor (&antcommProtocol:: SetMaxQueueTime,
202                                 &antcommProtocol:: GetMaxQueueTime),
203             MakeTimeChecker ())
204 .AddAttribute ("AllowedHelloLoss", "Number_of_hello_messages_which_may_be_loss_for_valid_link.",
205             UIntegerValue (2),
206             MakeUIntegerAccessor (&antcommProtocol::AllowedHelloLoss),
207             MakeUIntegerChecker<uint16_t> ())
208 .AddAttribute ("PheromoneDecayRate", "The_pheromone_decay_rate.",
209             DoubleValue (0.5),
210             MakeDoubleAccessor (&antcommProtocol::m_pheromoneDecayRate),
211             MakeDoubleChecker <double>())
212 .AddAttribute ("Alpha", "The_alpha.",
213             DoubleValue (0.33),
214             MakeDoubleAccessor(&antcommProtocol::m_alpha),
215             MakeDoubleChecker<double>())
216 .AddAttribute ("Beta", "The_beta.",
217             DoubleValue (0.33),
218             MakeDoubleAccessor (&antcommProtocol::m_beta),
219             MakeDoubleChecker < double >())
220 .AddAttribute ("Gamma", "The_gamma.",
221             DoubleValue (0.33),
222             MakeDoubleAccessor(&antcommProtocol::m_gamma),
223             MakeDoubleChecker<double>())
224 .AddAttribute ("EnableBroadcast", "Indicates_whether_a_broadcast_data_packets_forwarding_enable.
            ",
225             BooleanValue (true),
226             MakeBooleanAccessor (&antcommProtocol::SetBroadcastEnable,
227                                 &antcommProtocol::GetBroadcastEnable),
228             MakeBooleanChecker ())
229 .AddAttribute ("UniformRv", "Access_to_the_underlying_UniformRandomVariable",
230             StringValue ("ns3::UniformRandomVariable"),
231             MakePointerAccessor(&antcommProtocol::m_uniformRandomVariable),
232             MakePointerChecker<UniformRandomVariable>())
233 ;

```



```

234     return tid;
235 }
236
237 antcommProtocol::~antcommProtocol ()
238 {
239 }
240 void antcommProtocol::SetMaxQueueLen (uint32_t len)
241 {
242     MaxQueueLen = len;
243     m_queue.SetMaxQueueLen (len);
244 }
245
246 void antcommProtocol::SetMaxQueueTime (Time t)
247 {
248     MaxQueueTime = t;
249     m_queue.SetQueueTimeout (t);
250 }
251
252 void antcommProtocol::DoDispose ()
253 {
254     NS_LOG_FUNCTION (this);
255     m_ipv4 = 0;
256     for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::iterator iter = m_socketAddresses.begin
        (); iter != m_socketAddresses.end (); iter++)
        {
257         iter->first->Close ();
258     }
259     m_socketAddresses.clear ();
260     for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::iterator iter =
        m_socketSubnetBroadcastAddresses.begin (); iter != m_socketSubnetBroadcastAddresses.end ();
261         iter++)
        {
262             iter->first->Close ();
263         }
264     m_socketSubnetBroadcastAddresses.clear ();
265     Ipv4RoutingProtocol::DoDispose ();
266 }
267
268 void antcommProtocol::UpdateCurrentPhase (Phase type)
269 {
270     NS_LOG_FUNCTION (this);
271     m_currentPhase = type;
272 }
273
274 int64_t antcommProtocol::AssignStreams (int64_t stream)
275 {
276     NS_LOG_FUNCTION (this << stream);
277     m_uniformRandomVariable->SetStream (stream);
278     return 1;
279 }
280
281 void antcommProtocol::SetIpv4 (Ptr < Ipv4 > ipv4)
282 {
283     NS_LOG_FUNCTION (this);
284     NS_ASSERT (ipv4 != 0);
285     NS_ASSERT (m_ipv4 == 0);
286     m_ipv4 = ipv4;
287
288     // Create lo route. It is asserted that the only one interface up for now is loopback
289     NS_ASSERT (m_ipv4->GetNInterfaces () == 1 && m_ipv4->GetAddress (0,0).GetLocal () == Ipv4Address
290         ("127.0.0.1"));
291     m_lo = m_ipv4->GetNetDevice (0);
292     NS_ASSERT (m_lo != 0);
293     /** This is how every node has access to its own energy
294         through the ipv4 layer 3. */
295     Ptr <Node> node = m_ipv4->GetObject< Node > ();
296     Ptr<EnergySourceContainer> EnergySourceContainerOnNode = node->GetObject<EnergySourceContainer>
297         ();
298     Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (
299         EnergySourceContainerOnNode->Get(0));
300     // Remember lo route
301     antcommRoutingTableEntry rt (m_lo, /*dst= */ Ipv4Address::GetLoopback (), /*hops= */ 1,
302         m_pheromone, /*energy= */ basicSourcePtr->GetRemainingEnergy(), /*RSSI= */ 0, Ipv4Address::
303         GetLoopback (),
304         Ipv4InterfaceAddress (Ipv4Address::GetLoopback (), Ipv4Mask("255.0.0.0
305             ")),
306         /*lifetime= */ Simulator::GetMaximumSimulationTime ());
307     m_routingTable.AddRoute (rt);

```

```

305     Simulator::ScheduleNow (&antcommProtocol::Start, this);
306 }
307
308
309 void antcommProtocol::NotifyInterfaceUp (uint32_t i)
310 {
311     NS_LOG_DEBUG("NotifyInterfaceUp function");
312     NS_LOG_FUNCTION (this << m_ipv4->GetAddress (i, 0).GetLocal ());
313     Ptr <Ipv4L3Protocol> l3 = m_ipv4->GetObject <Ipv4L3Protocol> ();
314     if (l3->GetNAddresses (i) > 1)
315     {
316         NS_LOG_WARN ("antcomm_does_not_work_with_more_than_one_address_per_each_interface.");
317     }
318     Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
319     if (iface.GetLocal () == Ipv4Address ("127.0.0.1"))
320     {
321         return;
322     }
323
324     // Create a socket to listen only on this interface
325     Ptr < Socket > socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::GetTypeId
326         ());
327     NS_ASSERT (socket != 0);
328     socket->SetRecvCallback (MakeCallback (&antcommProtocol::Recvantcomm, this));
329     socket->Bind (InetSocketAddress (Ipv4Address::GetAny (), antcomm_PORT));
330     socket->BindToNetDevice (l3->GetNetDevice (i));
331     socket->SetAllowBroadcast (true);
332     socket->SetAttribute ("IpTtl", UintegerValue (1));
333     m_socketAddresses.insert (std::make_pair (socket, iface));
334
335     // create also a subnet broadcast socket
336     socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::GetTypeId ());
337     NS_ASSERT (socket != 0);
338     socket->SetRecvCallback (MakeCallback (&antcommProtocol::Recvantcomm, this));
339     socket->Bind (InetSocketAddress (iface.GetBroadcast (), antcomm_PORT));
340     socket->BindToNetDevice (l3->GetNetDevice (i));
341     socket->SetAllowBroadcast (true);
342     socket->SetAttribute ("IpTtl", UintegerValue (1));
343     m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));
344
345     // Add local broadcast record to the routing table
346     Ptr < NetDevice > dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()))
347     );
348     /** This is how every node has access to its own energy
349     through the ipv4 layer 3. */
350
351     Ptr <Node> node = m_ipv4->GetObject< Node > ();
352     Ptr<EnergySourceContainer> EnergySourceContainerOnNode = node->GetObject<EnergySourceContainer>
353     ();
354     Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (
355         EnergySourceContainerOnNode->Get(0));
356
357     antcommRoutingTableEntry rt (dev, /*dst= */ iface.GetBroadcast (),
358         /*hops= */ 1, m_pheromone, /*Energy=*/basicSourcePtr->
359             GetRemainingEnergy(), /*RSSI=*/ 0, /*Next Hop*/
360             iface.GetBroadcast (),
361             /*Interface*/ iface, /*lifetime= */
362             Simulator::GetMaximumSimulationTime ());
363
364     m_routingTable.AddRoute (rt);
365
366     if (l3->GetInterface (i)->GetArpCache ())
367     {
368         m_neighborTable.AddArpCache (l3->GetInterface (i)->GetArpCache ());
369     }
370     // Allow neighbor manager use this interface for layer 2 feedback if possible
371     Ptr < WifiNetDevice > wifi = dev->GetObject < WifiNetDevice > ();
372     if (wifi == 0)
373     {
374         return;
375     }
376     Ptr < WifiMac > mac = wifi->GetMac ();
377     if (mac == 0)
378     {
379         return;
380     }
381     mac->TraceConnectWithoutContext ("TxErrHeader", m_neighborTable.GetTxErrorCallback ());
382 }
383
384 void antcommProtocol::NotifyInterfaceDown (uint32_t i)
385 {
386     NS_LOG_DEBUG ("NotifyInterfaceDown function");
387     NS_LOG_FUNCTION (this << m_ipv4->GetAddress (i, 0).GetLocal ());
388     // Disable layer 2 link state monitoring (if possible)

```

```

379     Ptr <Ipv4L3Protocol> l3 = m_ipv4->GetObject <Ipv4L3Protocol>();
380     Ptr <NetDevice> dev = l3->GetNetDevice (i);
381     Ptr <WifiNetDevice> wifi = dev->GetObject <WifiNetDevice>();
382     if (wifi != 0)
383     {
384         Ptr <WifiMac> mac = wifi->GetMac ()->GetObject <AdhocWifiMac> ();
385         if (mac != 0)
386         {
387             mac->TraceDisconnectWithoutContext ("TxErrHeader", m_neighborTable.GetTxErrorCallback
388                 ());
389             m_neighborTable.DelArpCache (l3->GetInterface (i)->GetArpCache ());
390         }
391     }
392     // Close socket
393     Ptr < Socket > socket = FindSocketWithInterfaceAddress (m_ipv4->GetAddress (i, 0));
394     NS_ASSERT (socket);
395     socket->Close ();
396     m_socketAddresses.erase (socket);
397     // Close socket
398     socket = FindSubnetBroadcastSocketWithInterfaceAddress (m_ipv4->GetAddress (i, 0));
399     NS_ASSERT (socket);
400     socket->Close ();
401     m_socketSubnetBroadcastAddresses.erase (socket);
402     if (m_socketAddresses.empty ())
403     {
404         NS_LOG_LOGIC ("No antcomm interfaces");
405         m_helloTimer.Cancel ();
406         m_neighborTable.Clear ();
407         return;
408     }
409 }
410
411 void antcommProtocol::NotifyAddAddress (uint32_t i, Ipv4InterfaceAddress address)
412 {
413     NS_LOG_DEBUG ("NotifyAddAddress function");
414     NS_LOG_FUNCTION (this << "interface" << i << "address" << address);
415     Ptr < Ipv4L3Protocol > l3 = m_ipv4->GetObject < Ipv4L3Protocol > ();
416     if (l3->IsUp (i))
417         return;
418     if (l3->GetNAddresses (i) == 1)
419     {
420         Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
421         Ptr < Socket > socket = FindSocketWithInterfaceAddress (iface);
422         if (!socket)
423         {
424             if (iface.GetLocal () == Ipv4Address ("127.0.0.1"))
425                 return;
426             // Create a socket to listen only on this interface
427             Ptr < Socket > socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::
428                 GetTypeId ());
429             NS_ASSERT (socket != 0);
430             socket->SetRecvCallback (MakeCallback(&antcommProtocol::Recvantcomm, this));
431             socket->Bind (InetSocketAddress (iface.GetLocal (), antcomm_PORT));
432             socket->BindToNetDevice (l3->GetNetDevice (i));
433             socket->SetAllowBroadcast (true);
434             m_socketAddresses.insert (std::make_pair (socket, iface));
435
436             // create also a subnet directed broadcast socket
437             socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::GetTypeId ());
438             NS_ASSERT (socket != 0);
439             socket->SetRecvCallback (MakeCallback(&antcommProtocol::Recvantcomm, this));
440             socket->Bind (InetSocketAddress (iface.GetBroadcast (), antcomm_PORT));
441             socket->BindToNetDevice (l3->GetNetDevice (i));
442             socket->SetAllowBroadcast (true);
443             socket->SetAttribute ("IpTtl", UintegerValue (1));
444             m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));
445
446             // Add local broadcast record to the routing table
447             Ptr < NetDevice > dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.
448                 GetLocal ()));
449             /** This is how every node has access to its own energy
450             through the ipv4 layer 3. */
451             Ptr <Node> node = m_ipv4->GetObject< Node > ();
452             Ptr<EnergySourceContainer> EnergySourceContainerOnNode = node->GetObject<
453                 EnergySourceContainer> ();
454             Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (
455                 EnergySourceContainerOnNode->Get(0));

```

```

453         antcommRoutingTableEntry rt (dev, /*dst= */iface.GetBroadcast (), /*hops= */ 1,
454             m_pheromone, basicSourcePtr->GetRemainingEnergy(),
455             /*RSSI=*/ 0, iface.GetBroadcast (), iface, /*lifetime= */Simulator::
456                 GetMaximumSimulationTime ());
457     }
458     else
459     {
460         NS_LOG_LOGIC("AODV does not work with more than one address per interface. Ignore added
461             address");
462     }
463 }
464 void antcommProtocol::NotifyRemoveAddress (uint32_t i, Ipv4InterfaceAddress address)
465 {
466     NS_LOG_DEBUG ("NotifyRemoveAddress function");
467     NS_LOG_FUNCTION (this);
468     Ptr < Socket > socket = FindSocketWithInterfaceAddress (address);
469     if (socket)
470     {
471         socket->Close ();
472         m_socketAddresses.erase (socket);
473
474         Ptr < Socket > unicastSocket = FindSubnetBroadcastSocketWithInterfaceAddress (address);
475         if (unicastSocket)
476         {
477             unicastSocket->Close ();
478             m_socketAddresses.erase (unicastSocket);
479         }
480     }
481     Ptr < Ipv4L3Protocol > l3 = m_ipv4->GetObject < Ipv4L3Protocol > ();
482     if (l3->GetNAddresses (i))
483     {
484         Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
485         // Create a socket to listen only on this interface
486         Ptr <Socket> socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::
487             GetTypeId ());
488         NS_ASSERT (socket != 0);
489         socket->SetRecvCallback (MakeCallback(&antcommProtocol::Recvantcomm, this));
490
491         // Bind to any IP address so that broadcasts can be received
492         socket->Bind (InetSocketAddress (iface.GetLocal (), antcomm_PORT));
493         socket->BindToNetDevice (l3->GetNetDevice (i));
494         socket->SetAllowBroadcast (true);
495         socket->SetAttribute ("IpTtl", UintegerValue (1));
496         m_socketAddresses.insert (std::make_pair (socket, iface));
497
498         // create also a unicast socket
499         socket = Socket::CreateSocket (GetObject < Node > (), UdpSocketFactory::GetTypeId ());
500         NS_ASSERT (socket != 0);
501         socket->SetRecvCallback (MakeCallback(&antcommProtocol::Recvantcomm, this));
502         socket->Bind (InetSocketAddress (iface.GetBroadcast (), antcomm_PORT));
503         socket->BindToNetDevice (l3->GetNetDevice (i));
504         socket->SetAllowBroadcast (true);
505         socket->SetAttribute ("IpTtl", UintegerValue (1));
506         m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));
507
508         // Add local broadcast record to the routing table
509         Ptr <NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.
510             GetLocal ()));
511         /** This is how every node has access to its own energy
512             through the ipv4 layer 3. */
513         Ptr <Node> node = m_ipv4->GetObject< Node > ();
514         Ptr<EnergySourceContainer> EnergySourceContainerOnNode = node->GetObject<
515             EnergySourceContainer> ();
516         Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (
517             EnergySourceContainerOnNode->Get(0));
518
519         antcommRoutingTableEntry rt (dev, /*dst= */iface.GetBroadcast (), /*hops= */
520             1, m_pheromone, /*energy=*/basicSourcePtr
521                 ->GetRemainingEnergy(),
522             /*RSSI=*/ 0, iface.GetBroadcast (), iface,
523             /*lifetime= */Simulator::
524                 GetMaximumSimulationTime ());
525
526         m_routingTable.AddRoute (rt);
527     }
528     if (m_socketAddresses.empty ())
529     {
530         NS_LOG_LOGIC ("No antcomm interfaces");
531         m_helloTimer.Cancel ();
532     }
533 }

```

```

524     m_neighborTable.Clear ();
525     return;
526 }
527 }
528 else
529 {
530     NS_LOG_LOGIC("Remove address not participating in antcomm operation");
531 }
532 }
533
534
535 void antcommProtocol::PrintRoutingTable (Ptr <OutputStreamWrapper>stream) const
536 {
537     *stream->GetStream () << std::fixed << std::setprecision (10);
538     *stream->GetStream () << "Node:_" << m_ipv4->GetObject<Node>()->GetId () << "_Time:_" <<
        Simulator::Now ().GetSeconds () << "s_";
539     //m_routingTable.Print (stream);
540     m_neighborTable.PrintTable (stream);
541 }
542
543
544 void antcommProtocol::PrintInformation (Ptr <OutputStreamWrapper>stream) const
545 {
546     *stream->GetStream () << "Node:_" << m_ipv4->GetObject<Node>()->GetId ()
547         << "_Time:_" << Now ().As (Time::S)
548         << "_LocalTime:_" << GetObject<Node>()->GetLocalTime ().As (Time::S)
549         << "_antcommRoutingTable" << std::endl;
550
551     //m_routingTable.Print (stream);
552     m_neighborTable.PrintTable (stream);
553     *stream->GetStream () << std::endl;
554 }
555
556 Ptr <Socket> antcommProtocol::FindSocketWithInterfaceAddress (Ipv4InterfaceAddress addr) const
557 {
558     NS_LOG_FUNCTION (this << addr);
559     for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.
        begin (); j != m_socketAddresses.end (); ++j)
560     {
561         Ptr < Socket > socket = j->first;
562         Ipv4InterfaceAddress iface = j->second;
563         if (iface == addr)
564             return socket;
565     }
566     Ptr < Socket > socket;
567     return socket;
568 }
569
570 Ptr <Socket> antcommProtocol::FindSubnetBroadcastSocketWithInterfaceAddress (Ipv4InterfaceAddress
    addr) const
571 {
572     NS_LOG_FUNCTION (this << addr);
573     for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j =
        m_socketSubnetBroadcastAddresses.begin ();
574         j != m_socketSubnetBroadcastAddresses.end (); ++j)
575     {
576         Ptr < Socket > socket = j->first;
577         Ipv4InterfaceAddress iface = j->second;
578         if (iface == addr)
579             return socket;
580     }
581     Ptr < Socket > socket;
582     return socket;
583 }
584
585 //Modified RouteOutput function
586 Ptr <Ipv4Route> antcommProtocol::RouteOutput (Ptr <Packet> p, const Ipv4Header &header,
        Ptr < NetDevice > oif, Socket::
        SocketErrno & sockerr)
587 {
588     NS_LOG_DEBUG ("This is RouteOutput functionality");
589     NS_LOG_FUNCTION (this << header << (oif ? oif->GetIfIndex () : 0));
590     if (!p)
591     {
592         NS_LOG_DEBUG ("Packet is 0");
593         return LoopbackRoute (header, oif); // later
594     }
595     if (m_socketAddresses.empty ())
596     {
597         sockerr = Socket::ERROR_NOROUTETOHOST;
598         NS_LOG_LOGIC ("No antcomm interfaces");

```

```

599         Ptr <Ipv4Route> route;
600         return route;
601     }
602     sockerr = Socket::ERROR_NOTERROR;
603     Ptr < Ipv4Route > route;
604     Ipv4Address dst = header.GetDestination ();
605
606     /*
607         check a valid route with the destination address
608         The function looks for the destination address. If there are multiple neighbors,
609         it selects one of them by considering pheromone, energy, hop count, etc. ,
610         and return the result as rt as if LookupValidRoute(dst, rt) function does in aodv
611
612     */
613
614     antcommRoutingTableEntry rt;
615     if (m_routingTable.LookupRoute2 (dst, rt))
616     {
617         route = rt.GetRoute ();
618         NS_ASSERT (route != 0);
619         NS_LOG_DEBUG ("The destination through route is " << dst );
620         NS_LOG_DEBUG ("Exist route to " << route->GetDestination () << " from interface " << route->
        GetSource ());
621         if (oif != 0 && route->GetOutputDevice () != oif)
622         {
623             NS_LOG_DEBUG ("Output device doesn't match. Dropped.");
624             sockerr = Socket::ERROR_NOROUTETOHOST;
625             return Ptr < Ipv4Route > ();
626         }
627         UpdateRouteLifeTime (dst, ActiveRouteTimeout);
628         UpdateRouteLifeTime (route->GetGateway (), ActiveRouteTimeout);
629         return route;
630     }
631     // Valid route not found, in this case we return loopback.
632     // Actual route request will be deferred until packet will be fully formed,
633     // routed to loopback, received from loopback and passed to RouteInput (see below)
634     uint32_t iif = (oif ? m_ipv4->GetInterfaceForDevice (oif) : -1);
635     DeferredRouteOutputTag tag (iif);
636     NS_LOG_DEBUG ("Valid Route not found");
637     if (!p->PeekPacketTag (tag))
638     {
639         p->AddPacketTag (tag);
640     }
641     return LoopbackRoute (header, oif);
642 }
643
644
645 Ptr <Ipv4Route> antcommProtocol::LoopbackRoute (const Ipv4Header & hdr, Ptr <NetDevice> oif) const
646 {
647     NS_LOG_FUNCTION (this << hdr);
648     NS_ASSERT (m_lo != 0);
649     Ptr < Ipv4Route > rt = Create < Ipv4Route > ();
650     rt->SetDestination (hdr.GetDestination ());
651     std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.begin ()
652     ;
653     if (oif)
654     {
655         // Iterate to find an address on the oif device
656         for (j = m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
657         {
658             Ipv4Address addr = j->second.GetLocal ();
659             int32_t interface = m_ipv4->GetInterfaceForAddress (addr);
660             if (oif == m_ipv4->GetNetDevice (static_cast < uint32_t > (interface)))
661             {
662                 rt->SetSource (addr);
663                 break;
664             }
665         }
666     }
667     else
668     {
669         rt->SetSource (j->second.GetLocal ());
670     }
671     NS_ASSERT_MSG (rt->GetSource () != Ipv4Address (), "Valid antcomm source address not found");
672     rt->SetGateway (Ipv4Address ("127.0.0.1"));
673     rt->SetOutputDevice (m_lo);
674     return rt;
675 }
676

```

```

677
678 void antcommProtocol::DeferredRouteOutput (Ptr < const Packet > p, const Ipv4Header & header,
679                                           UnicastForwardCallback ucb,
680                                           ErrorCallback ecb)
681 {
682     NS_LOG_FUNCTION (this << p << header);
683     NS_ASSERT (p != 0 && p != Ptr < Packet > ());
684
685     QueueEntry newEntry (p, header, ucb, ecb);
686     bool result = m_queue.Enqueue (newEntry);
687     if (result)
688     {
689         NS_LOG_LOGIC ("Add_packet" << p->GetUid () << "to_queue.Protocol" << (uint16_t)header.
690                       GetProtocol ());
691         antcommRoutingTableEntry rt;
692         bool result = m_routingTable.LookupRoute2 (header.GetDestination (), rt);
693         if (!result)
694         {
695             NS_LOG_LOGIC ("Send_new_RREQ_for_outbound_packet" << header.GetDestination ());
696             SendEXPL (header.GetDestination ());
697         }
698     }
699 }
700
701 bool antcommProtocol::IsMyOwnAddress (Ipv4Address src)
702 {
703     NS_LOG_FUNCTION (this << src);
704     for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j =
705           m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
706     {
707         Ipv4InterfaceAddress iface = j->second;
708         if (src == iface.GetLocal ())
709         {
710             return true;
711         }
712     }
713     return false;
714 }
715
716 bool antcommProtocol::RouteInput (Ptr < const Packet > p, const Ipv4Header &header,
717                                   Ptr < const NetDevice > idev,
718                                   UnicastForwardCallback ucb,
719                                   MulticastForwardCallback mcb,
720                                   LocalDeliverCallback lcb,
721                                   ErrorCallback ecb)
722 {
723     NS_LOG_FUNCTION (this << p->GetUid () << header.GetDestination () << idev->GetAddress ());
724     NS_LOG_DEBUG ("antcomm_node" << m_ipv4->GetObject<Node> ()->GetId () << "from RouteInput"
725                  << received_a_packet);
726     if (m_socketAddresses.empty ())
727     {
728         NS_LOG_LOGIC ("No antcomm interfaces");
729         return false;
730     }
731
732     NS_ASSERT (m_ipv4 != 0);
733     NS_ASSERT (p != 0);
734     // Check if input device supports IP
735     NS_ASSERT (m_ipv4->GetInterfaceForDevice (idev) >= 0);
736     int32_t iif = m_ipv4->GetInterfaceForDevice (idev);
737
738     Ipv4Address dst = header.GetDestination ();
739     Ipv4Address origin = header.GetSource ();
740
741     NS_LOG_DEBUG ("dst through RouteInput is" << dst);
742     NS_LOG_DEBUG ("origin through RouteInput is" << origin);
743
744     // Deferred route request
745     if (idev == m_lo)
746     {
747         DeferredRouteOutputTag tag;
748         if (p->PeekPacketTag (tag))
749         {
750             DeferredRouteOutput (p, header, ucb, ecb);
751             return true;
752         }
753     }
754
755     // Duplicate of own packet
756     if (IsMyOwnAddress (origin))

```

```

752     {
753     return true;
754     }
755 if (dst.IsMulticast ())
756     {
757     return false;
758     }
759
760 for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j =
761     m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
762     {
763     Ipv4InterfaceAddress iface = j->second;
764     if (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()) == iif)
765     {
766     if (dst == iface.GetBroadcast () || dst.IsBroadcast ())
767     {
768     if (m_dpd.IsDuplicate (p, header))
769     {
770     NS_LOG_DEBUG ("Duplicated_packets_" << p->GetUid() << "_from_" << origin
771     << "_Drop_");
772     return true;
773     }
774     UpdateRouteLifeTime (origin, ActiveRouteTimeout);
775     Ptr < Packet > packet = p->Copy ();
776     if (lcb.IsNull () == false)
777     {
778     NS_LOG_LOGIC ("Broadcast_local_delivery_to_" << iface.GetLocal ());
779     lcb (p, header, iif);
780     }
781     else
782     {
783     NS_LOG_ERROR("Unable_to_deliver_packet_locally_due_to_null_callback"<< p->GetUid () << "
784     _from_" << origin);
785     ecb (p, header, Socket::ERROR_NOROUTETOHOST);
786     }
787     if (!EnableBroadcast)
788     {
789     return true;
790     }
791     if (header.GetTtl () > 1)
792     {
793     NS_LOG_LOGIC ("Forward_broadcast_TTL_" << (uint16_t)header.GetTtl ());
794     antcommRoutingTableEntry toBroadcast;
795     if (m_routingTable.LookupRoute2 (dst, toBroadcast))
796     {
797     Ptr < Ipv4Route > route = toBroadcast.GetRoute ();
798     ucb (route, packet, header);
799     }
800     else
801     {
802     NS_LOG_DEBUG ("No_route_to_forward_broadcast.Drop_packet_" << p->GetUid ());
803     }
804     }
805     else
806     {
807     NS_LOG_DEBUG ("TTL_exceeded.Drop_packet_" << p->GetUid ());
808     }
809     return true;
810     }
811     }
812 // Unicast local delivery
813 if (m_ipv4->IsDestinationAddress (dst, iif))
814     {
815     double x = 0;
816     SnrTag tag;
817     if (p->PeekPacketTag(tag))
818     {
819     NS_LOG_DEBUG ("the_snr_of_the_packet_is_" << tag.Get() << "dBm");
820     x = tag.Get();
821     }
822     UpdateRouteLifeTime (origin, ActiveRouteTimeout);
823     antcommRoutingTableEntry toOrigin;
824     if (m_routingTable.LookupRoute2 (origin, toOrigin))
825     {
826     UpdateRouteLifeTime (toOrigin.GetNextHop (), ActiveRouteTimeout);
827     double rssi = m_neighborTable.GetRssi (toOrigin.GetNextHop ());
828     double energy = m_neighborTable.GetEnergy (toOrigin.GetNextHop ());

```



```

830     m_neighborTable.Update (toOrigin.GetNextHop (), rssi, energy, ActiveRouteTimeout);
831 }
832 if (lcb.IsNull () == false)
833 {
834     NS_LOG_LOGIC ("Unicast_local_delivery_to_" << dst);
835     lcb (p, header, iif);
836 }
837 else
838 {
839     NS_LOG_ERROR ("Unable_to_deliver_packet_locally_due_to_null_callback_" << p->GetUid () << "_"
840                 from_" << origin);
841     ecb (p, header, Socket::ERROR_NOROUTETOHOST);
842 }
843 return true;
844 }
845 // Check if input device supports IP forwarding
846 if (m_ipv4->IsForwarding (iif) == false)
847 {
848     NS_LOG_LOGIC ("Forwarding_disabled_for_this_interface");
849     ecb (p, header, Socket::ERROR_NOROUTETOHOST);
850     return true;
851 }
852
853 // Forwarding
854 return Forwarding (p, header, ucb, ecb);
855 }
856
857 bool antcommProtocol::Forwarding (Ptr < const Packet > p, const Ipv4Header & header,
858                                 UnicastForwardCallback ucb, ErrorCallback ecb)
859 {
860     NS_LOG_DEBUG ("Forwarding_function_");
861     NS_LOG_FUNCTION (this);
862     Ipv4Address dst = header.GetDestination ();
863     Ipv4Address origin = header.GetSource ();
864
865     antcommRoutingTableEntry toDst;
866     if (m_routingTable.LookupRoute2 (dst, toDst))
867     {
868         if (toDst.GetPheromone() < 0.5)
869             toDst.SetPheromone(INITIAL_PHEROMONE);
870         Ptr < Ipv4Route > route = toDst.GetRoute ();
871         NS_LOG_LOGIC (route->GetSource () << "_forwarding_to_" << dst << "_from_" << origin << "_
872                     packet_" << p->GetUid ());
873
874         toDst.SetPheromone (GetPheromone() + 0.5);
875         UpdateRouteLifeTime (origin, ActiveRouteTimeout);
876         UpdateRouteLifeTime (dst, ActiveRouteTimeout);
877         UpdateRouteLifeTime (route->GetGateway (), ActiveRouteTimeout);
878
879         antcommRoutingTableEntry toOrigin;
880         m_routingTable.LookupRoute2 (origin, toOrigin);
881         UpdateRouteLifeTime (toOrigin.GetNextHop (), ActiveRouteTimeout);
882
883         m_neighborTable.Update (route->GetGateway (), m_neighborTable.GetRssi (route->GetGateway ()),
884                               m_neighborTable.GetEnergy (route->GetGateway ()), ActiveRouteTimeout);
885         m_neighborTable.Update (toOrigin.GetNextHop (), m_neighborTable.GetRssi (toOrigin.GetNextHop
886                                     ()), m_neighborTable.GetEnergy (toOrigin.GetNextHop ()), ActiveRouteTimeout);
887
888         ucb (route, p, header);
889         return true;
890     }
891     else
892     {
893         NS_LOG_DEBUG ("Drop_packet_" << p->GetUid () << "_because_no_route_to_forward_it.");
894         return false;
895     }
896 }
897 //return false;
898
899 void antcommProtocol::Start ()
900 {
901     NS_LOG_FUNCTION (this);
902     //m_explorationTimer.SetFunction (&antcommProtocol::SendHello, this);
903     //m_explorationTimer.Schedule (Seconds (m_explorationTime));
904     m_neighborTable.ScheduleTimer ();
905     m_forwardTimer.SetFunction (&antcommProtocol::ExplRateLimitTimerExpire, this);
906     m_forwardTimer.Schedule (Seconds (1));
907     //m_backwardTimer.SetFunction (&antcommProtocol::SendREPA, this);

```

```

906     //m_backwardTimer.Schedule (Seconds (1));
907 }
908
909 void antcommProtocol::SendTo (Ptr <Socket> socket, Ptr <Packet> packet, Ipv4Address destination)
910 {
911     NS_LOG_FUNCTION (this);
912
913     //std::cout << "Sent a data packet " << this << "\n";
914     socket->SendTo (packet, 0, InetSocketAddress (destination, antcomm_PORT));
915 }
916
917 void antcommProtocol::Recvantcomm (Ptr < Socket > socket)
918 {
919     Address sourceAddress;
920     Ptr < Packet > packet = socket->RecvFrom (sourceAddress);
921     InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (sourceAddress);
922     Ipv4Address sender = inetSourceAddr.GetIpv4 ();
923     Ipv4Address my;
924     if (m_socketAddresses.find (socket) != m_socketAddresses.end ())
925     {
926         my = m_socketAddresses[socket].GetLocal ();
927     }
928     else if (m_socketSubnetBroadcastAddresses.find (socket) != m_socketSubnetBroadcastAddresses.end
929              ())
930     {
931         my = m_socketSubnetBroadcastAddresses[socket].GetLocal ();
932     }
933     else
934     {
935         NS_ASSERT_MSG (false, "Received a packet from an unknown socket");
936         NS_LOG_DEBUG ("antcomm_node_" << m_ipv4->GetObject<Node> ()->GetId () << " received a antcomm_
937             packet from " << sender << " to " << my);
938
939         TypeHeader tHeader (EXPL);
940         packet->RemoveHeader (tHeader);
941         //uint16_t myNodeID = m_ipv4->GetObject<Node> ()->GetId (); //uint16_t
942         if (!tHeader.IsValid ())
943         {
944             NS_LOG_DEBUG ("antcomm_message_" << packet->GetUid () << " with unknown type received: " <<
945                 tHeader.Get () << ". Drop");
946             return; // drop
947         }
948         //if (IsMyOwnAddress("10.0.0.5")) {
949         //NS_LOG("lsdfkjalsdfkj");
950         //}
951         if (tHeader.Get () == EXPL)
952             NS_LOG_INFO ("The packet was tagged as EXPL and then it was received");
953         if (tHeader.Get () == REPA && IsMyOwnAddress ("10.0.0.5"))
954             NS_LOG_INFO ("The packet was tagged as REPA and hopefully it should be received");
955
956         switch (tHeader.Get ())
957         {
958             {
959                 case HELLO:
960                 {
961                     RecvHELLO (packet, my, sender);
962                     //MakeRoutingTable (my);
963                     break;
964                 }
965                 case EXPL:
966                 {
967                     RecvEXPL (packet, my, sender);
968                     break;
969                 }
970                 case REPA:
971                 {
972                     RecvREPA (packet, my, sender);
973                     break;
974                 }
975                 case RE_REPA:
976                 {
977                     antcommREREPAHeader headerRerepa;
978                     packet->RemoveHeader (headerRerepa);
979                     //RecvREREPA (headerRerepa, my, headerRerepa.GetOrigin (), sender);
980                     break;
981                 }
982             }

```

```

983     }
984 }
985
986 bool antcommProtocol::UpdateRouteLifeTime (Ipv4Address addr, Time lifetime)
987 {
988     NS_LOG_FUNCTION (this << addr << lifetime);
989     antcommRoutingTableEntry rt;
990     if (m_routingTable.LookupRoute2 (addr, rt))
991     {
992         //NS_LOG_DEBUG ("Updating VALID route");
993         rt.SetUpdatedTime (std::max (lifetime, rt.GetUpdatedTime ()));
994         m_routingTable.Update (rt); // update
995         return true;
996     }
997     return false;
998 }
999
1000 void antcommProtocol::MakeRoutingTable (Ipv4Address origin, Ipv4Address my, Ipv4Address src) //
1001     modifying right now
1002 {
1003     int degree = 0;
1004     degree = m_neighborTable.Degree();
1005     antcommRoutingTableEntry toOrigin;
1006
1007     while (degree-- > 0)
1008     {
1009         //std::cout << "***" << my << " " << degree+1 << ": " << m_neighborTable.GetAddress (degree) <<
1010             std::endl;
1011         if (src.IsEqual (m_neighborTable.GetAddress (degree)))
1012         {
1013             if (m_routingTable.LookupRoute (origin, m_neighborTable.GetAddress (degree), toOrigin) == 0)
1014             {
1015                 Ptr <NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (my));
1016                 antcommRoutingTableEntry newEntry (dev, origin, 1, 0.2, m_neighborTable.GetEnergy (src),
1017                     m_neighborTable.GetRssi (src), src,
1018                     m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (my), 0),
1019                     Time ((2* NetTraversalTime - 2 * NodeTraversalTime)));
1020                 m_routingTable.AddRoute (newEntry);
1021             }
1022         }
1023     }
1024
1025     //return;
1026
1027     //std::cout << "*** while done\n";
1028 }
1029
1030 /// Receive HELLO
1031 void antcommProtocol::RecvHELLO (Ptr < Packet > p, Ipv4Address my, Ipv4Address sender)
1032 {
1033     antcommHelloHeader headerHello;
1034     p->RemoveHeader (headerHello);
1035     NS_LOG_INFO ("HELLO_ from_ " << sender << ",_ to_ " << headerHello.GetOrigin () << ",_ to_ " <<
1036         my);
1037     headerHello.SetTimeToLive (headerHello.GetTimeToLive () - 1);
1038     headerHello.SetHopCount (headerHello.GetHopCount () + 1);
1039     double x = 0;
1040     double distance;
1041     double probability;
1042     double goodness_rssi = 0;
1043     //double rssi_previous = 0;
1044     SnrTag tag;
1045     if (p->PeekPacketTag(tag))
1046     {
1047         NS_LOG_DEBUG ("the_snr_of_the_packet_is_ " << tag.Get() << "dBm");
1048         x = tag.Get();
1049     }
1050     distance = exp((x - 117.17)/(-28.35));
1051     if (distance/50.92 < 1)
1052     {
1053         probability = 1- 0.5 * pow ((distance/50.92), 4);
1054     }
1055     else
1056     {
1057         probability = 0.5 * pow(((2*50.92-distance)/50.92), 4);
1058     }
1059     goodness_rssi = distance * pow (probability, 2) / (50.92 * (1 + probability));

```

```

1059 //std::cout << "The residual energy of node in hello function is " << headerHello.
1060 GetResidualEnergy () << std::endl;
1061 /* Higher RSSI value (meaning higher number of hops) and lower RSSI value (higher packet drop
1062 chances)
1063 is not good for our probabilistic system. Hence our goodness_rssi should follow that curve that
1064 has minima on extreme ends and maxima in a middle. The curve  $a * e^{-x}$  is goodness rssi where "
1065 a" is
1066 any constant and x is rssi value from physical layer. In our case I have taken "a" as 7
1067 */
1068 //goodness_rssi = 7*x*exp(-x);
1069 //m_rssi = goodness_rssi;
1070
1071 /* std::cout << " receiving hello message\n" <<
1072 " from "<<(Ipv4Address)sender<<std::endl<<
1073 " origin "<<(Ipv4Address)headerHello.GetOrigin()<<std::endl<<
1074 " to "<<(Ipv4Address)my<<std::endl; */
1075 m_neighborTable.Update (sender, goodness_rssi, headerHello.GetResidualEnergy(), Time (
1076 AllowedHelloLoss * HelloInterval));
1077
1078 }
1079
1080 void antcommProtocol::RecvEXPL (Ptr <Packet> p, Ipv4Address my, Ipv4Address src)
1081 {
1082     NS_LOG_FUNCTION (this);
1083     antcommEXPLHeader headerEXPL;
1084     p->RemoveHeader (headerEXPL);
1085
1086     /* std::cout << " I am receiving Request message\n" <<
1087     " from "<<(Ipv4Address)sender<<std::endl<<
1088     " originator "<<(Ipv4Address)headerEXPL.GetOrigin()<<std::endl<<
1089     " My IP address is "<<(Ipv4Address)my<<std::endl<<
1090     " The intended destination is "<<(Ipv4Address)headerEXPL.GetDst()<<std::endl; */
1091
1092     NS_LOG_INFO ("Request_EXPL_from_ " << src << ",_origin_" << headerEXPL.GetOrigin () << ",_to_"
1093 << my);
1094     Ipv4Address destination = headerEXPL.GetDst ();
1095     std::cout << destination << std::endl; // 10.0.0.5
1096     Ipv4Address origin = headerEXPL.GetOrigin ();
1097     std::cout << origin << std::endl; // 10.0.0.1
1098
1099     MakeRoutingTable (origin, my, src);
1100
1101     uint32_t id = headerEXPL.GetId ();
1102     if (m_rreqIdCache.IsDuplicate (origin, id))
1103     {
1104         NS_LOG_DEBUG ("Ignoring_request_message_due_to_duplication");
1105         return;
1106     }
1107
1108     //Increment RREQ hop Count
1109     uint16_t hop = headerEXPL.GetHopCount () + 1;
1110     headerEXPL.SetHopCount (hop);
1111     headerEXPL.SetTimeToLive (headerEXPL.GetTimeToLive () - 1);
1112     //std::cout<<"I am this hop "<<(double)headerEXPL.GetHopCount() << " away from "<<(Ipv4Address)
1113     headerEXPL.GetOrigin()<<std::endl;
1114     antcommRoutingTableEntry toOrigin; // Discuss this methodology, AODV has the same
1115     if (m_routingTable.LookupRoute (origin, src, toOrigin))
1116     //If the routing table entry exists, set the parameters and update the routing table
1117     {
1118         if(toOrigin.GetPheromone() < 0.5)
1119             toOrigin.SetPheromone(INITIAL_PHEROMONE);
1120         toOrigin.SetPheromone (GetPheromone() + 0.2);
1121         toOrigin.SetHopCount (hop);
1122         toOrigin.SetRssi (m_neighborTable.GetRssi (src));
1123         toOrigin.SetNextHop (src);
1124         toOrigin.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress(my)));
1125         toOrigin.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (my), 0));
1126         toOrigin.SetResidualEnergy (m_neighborTable.GetEnergy (src));
1127         toOrigin.SetUpdatedTime (std::max (Time (2 * NetTraversalTime - 2 * hop * NodeTraversalTime),
1128             toOrigin.GetUpdatedTime ()));
1129         m_routingTable.Update (toOrigin); // update
1130     }
1131     else // otherwise make a new entry and add it in routing table
1132     {
1133         Ptr <NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (my));
1134
1135         antcommRoutingTableEntry newEntry (dev, origin, hop, INITIAL_PHEROMONE, m_neighborTable.
1136             GetEnergy (src),
1137
1138             m_neighborTable.GetRssi (src), src,

```

```

1130                                     m_ipv4->GetAddress (m_ipv4->
1131                                     GetInterfaceForAddress(my), 0),
                                     Time ((2 * NetTraversalTime - 2 *
                                     hop * NodeTraversalTime));
1132                                     //std::cout << "Adding route (" << src << ", " << headerEXPL.GetOrigin() <<")\n";
1133                                     NS_LOG_INFO ("Adding route (" << src << ", " << origin << "\n");
1134                                     m_routingTable.AddRoute (newEntry);
1135                                     }
1136
1137                                     if (IsMyOwnAddress (headerEXPL.GetDst ()))
1138                                     {
1139                                         m_routingTable.LookupRoute2 (origin, toOrigin);
1140                                         NS_LOG_DEBUG ("Send reply since I am the destination");
1141                                         SendREPA (headerEXPL, toOrigin);
1142                                         return;
1143                                     }
1144                                     //else scan the entries from my socket
1145                                     for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.
                                     begin (); j != m_socketAddresses.end (); ++j)
1146                                     {
1147                                         Ptr < Socket > socket = j->first;
1148                                         Ipv4InterfaceAddress iface = j->second;
1149                                         Ptr < Packet > packet = Create < Packet > ();
1150                                         packet->AddHeader (headerEXPL);
1151                                         TypeHeader tHeader (EXPL);
1152                                         packet->AddHeader (tHeader);
1153                                         // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
1154                                         Ipv4Address destination;
1155                                         if (iface.GetMask () == Ipv4Mask::GetOnes ())
1156                                         {
1157                                             destination = Ipv4Address ("255.255.255.255");
1158                                         }
1159                                         else
1160                                         {
1161                                             destination = iface.GetBroadcast ();
1162                                         }
1163
1164                                         m_lastBcastTime = Simulator::Now ();
1165                                         Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
1166                                             &antcommProtocol::SendTo, this, socket, packet, destination);
1167                                     }
1168
1169                                     }
1170
1171                                     }
1172                                     void antcommProtocol::SendReplyByIntermediateNode (antcommRoutingTableEntry & toDst,
1173                                     antcommRoutingTableEntry & toOrigin)
1174                                     {
1175                                         NS_LOG_FUNCTION (this);
1176                                         antcommREPAHeader headerRepa (/hops=*/ toDst.GetHopCount (), /*Time To Live*/ 255, /*origin=*/
                                         toOrigin.GetDestination (), /*dst=*/ toDst.GetDestination ());
1177
1178                                         m_routingTable.Update (toDst);
1179                                         m_routingTable.Update (toOrigin);
1180
1181                                         Ptr<Packet> packet = Create<Packet> ();
1182                                         packet->AddHeader (headerRepa);
1183                                         TypeHeader tHeader (REPA);
1184                                         packet->AddHeader (tHeader);
1185                                         Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
1186                                         NS_ASSERT (socket);
1187                                         socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), antcomm_PORT));
1188
1189                                         // Generating gratuitous RREPs (add later if required, otherwise no need)
1190                                     }
1191
1192                                     /// Receive REPA
1193                                     void antcommProtocol::RecvREPA (Ptr < Packet > p, Ipv4Address my, Ipv4Address sender)
1194                                     {
1195                                         NS_LOG_FUNCTION (this << "neighbor is" << sender);
1196                                         antcommREPAHeader headerRepa;
1197                                         p->RemoveHeader (headerRepa);
1198
1199                                         NS_LOG_INFO ("Reply from" << sender << ", origin" << headerRepa.GetDst () << ", to" << my);
1200                                         //std::cout<<" I have received the reply packet" << std::endl;
1201                                         Ipv4Address dst = headerRepa.GetDst ();
1202
1203                                         std::cout << "destination through REPA header is" << dst << std::endl; // 10.0.0.5
1204

```

```

1205     std::cout << "origin_through_REPA_header_is_" << headerRepa.GetOrigin() << std::endl; //10.0.0.1
1206
1207     uint16_t hop = headerRepa.GetHopCount () + 1;
1208     headerRepa.SetHopCount (hop);
1209     headerRepa.SetTimeToLive (headerRepa.GetTimeToLive () - 1);
1210     /*Each node receiving REPA will check the routing table entry
1211     for destination (generator of EXPL message). If the entry exist,
1212     update the entries otherwise add */
1213     Ptr < NetDevice > dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (my));
1214
1215     antcommRoutingTableEntry newEntry (dev, dst, hop, INITIAL_PHEROMONE, m_neighborTable.GetEnergy (
1216                                     sender),
                                     m_neighborTable.GetRssi (sender),
                                     sender, m_ipv4->GetAddress (
                                     m_ipv4->GetInterfaceForAddress(my
                                     ), 0), Simulator::Now ());
1217
1218     antcommRoutingTableEntry toDst;
1219     if (m_routingTable.LookupRoute (dst, sender, toDst))
1220     {
1221         if (toDst.GetPheromone() < 0.5)
1222             toDst.SetPheromone(INITIAL_PHEROMONE);
1223         toDst.SetPheromone (GetPheromone() + 0.2);
1224         toDst.SetHopCount (hop);
1225         toDst.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (my)));
1226         toDst.SetNextHop (sender);
1227         toDst.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (my), 0));
1228         toDst.SetResidualEnergy (m_neighborTable.GetEnergy (sender));
1229         toDst.SetRssi (m_neighborTable.GetRssi (sender));
1230         m_routingTable.Update (toDst); // update
1231     }
1232     else
1233         m_routingTable.AddRoute (newEntry);
1234
1235     if (IsMyOwnAddress (headerRepa.GetOrigin ()))
1236     {
1237         std::cout << headerRepa.GetOrigin() << std::endl;
1238         NS_LOG_DEBUG ("Route_establishment_phase_completed_since_I_am_the_source_and_have_received_
1239                     REPA_packet");
1240         m_routingTable.LookupRoute2 (dst, toDst);
1241         SendPacketFromQueue (dst, toDst.GetRoute ());
1242         return;
1243     }
1244     //SendREREPA();
1245
1246     antcommRoutingTableEntry toOrigin;
1247     if (m_routingTable.LookupRoute2 (headerRepa.GetOrigin (), toOrigin))
1248     {
1249         toOrigin.SetUpdatedTime (std::max (ActiveRouteTimeout, toOrigin.GetUpdatedTime ()));
1250         m_routingTable.Update (toOrigin); // update
1251     }
1252     else
1253         return;
1254     Ptr < Packet > packet = Create < Packet > ();
1255     packet->AddHeader (headerRepa);
1256     TypeHeader tHeader (REPA);
1257     packet->AddHeader (tHeader);
1258     Ptr < Socket > socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
1259     NS_ASSERT (socket);
1260     socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), antcomm_PORT));
1261 }
1262
1263 void antcommProtocol::SendPacketFromQueue (Ipv4Address dst, Ptr <Ipv4Route> route)
1264 {
1265     NS_LOG_FUNCTION (this);
1266     std::cout << "test_case_to_see_whether_packets_from_buffer_are_being_sent_or_not" << std::endl;
1267     QueueEntry queueEntry;
1268     while (m_queue.Dequeue (dst, queueEntry))
1269     {
1270         DeferredRouteOutputTag tag;
1271         Ptr <Packet> p = ConstCast <Packet> (queueEntry.GetPacket ());
1272         if (p->RemovePacketTag (tag) &&
1273             tag.GetInterface () != -1 &&
1274             tag.GetInterface () != m_ipv4->GetInterfaceForDevice (route->GetOutputDevice ()))
1275         {
1276             NS_LOG_DEBUG ("Output_device_doesn't_match..Dropped.");
1277             return;
1278         }
1279         UnicastForwardCallback ucb = queueEntry.GetUnicastForwardCallback ();
1280         Ipv4Header header = queueEntry.GetIpv4Header ();
1281         header.SetSource (route->GetSource ());

```

```

1280     header.SetTtl (header.GetTtl () + 1); // compensate extra TTL decrement by fake loopback routing
1281     ucb (route, p, header);
1282     }
1283 }
1284
1285 void antcommProtocol::SendEXPL (Ipv4Address dst)
1286 {
1287     if (m_explCount == EXPLRateLimit)
1288     {
1289         Simulator::Schedule (m_forwardTimer.GetDelayLeft () + MicroSeconds (100),
1290                             &antcommProtocol::SendEXPL, this, dst);
1291         return;
1292     }
1293     else
1294     {
1295         m_explCount++;
1296         //Create RREQ header
1297         antcommEXPLHeader headerEXPL;
1298         headerEXPL.SetDst (dst);
1299         antcommRoutingTableEntry rt;
1300         if (m_routingTable.LookupRoute2 (dst, rt))
1301         {
1302             headerEXPL.SetHopCount (rt.GetHopCount ());
1303             m_routingTable.Update (rt); // update
1304         }
1305         else
1306         {
1307             Ptr < NetDevice > dev = 0;
1308             Ptr <Node> node = m_ipv4->GetObject< Node > ();
1309             Ptr<EnergySourceContainer> EnergySourceContainerOnNode = node->GetObject<
1310                 EnergySourceContainer> ();
1311             Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (
1312                 EnergySourceContainerOnNode->Get(0));
1313             antcommRoutingTableEntry newEntry (dev, dst, 1/*0*/, GetPheromone(), basicSourcePtr->
1314                 GetRemainingEnergy(), 0, Ipv4Address (), Ipv4InterfaceAddress (), Seconds (0));
1315             m_routingTable.AddRoute (newEntry);
1316         }
1317
1318         //UpdateCurrentPhase (EXPLORATION);
1319         m_requestId++;
1320         headerEXPL.SetId (m_requestId);
1321         headerEXPL.SetHopCount (0);
1322         headerEXPL.SetTimeToLive (255);
1323         for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.
1324             begin (); j != m_socketAddresses.end (); ++j)
1325         {
1326             Ptr < Socket > socket = j->first;
1327             Ipv4InterfaceAddress iface = j->second;
1328
1329             headerEXPL.SetOrigin (iface.GetLocal ());
1330             m_rreqIdCache.IsDuplicate (iface.GetLocal (), m_requestId);
1331             Ptr < Packet > packet = Create < Packet > ();
1332             packet->AddHeader (headerEXPL);
1333             TypeHeader tHeader (EXPL);
1334             packet->AddHeader (tHeader);
1335             Ipv4Address destination;
1336             if (iface.GetMask () == Ipv4Mask::GetOnes ())
1337             {
1338                 destination = Ipv4Address ("255.255.255.255");
1339             }
1340             else
1341             {
1342                 destination = iface.GetBroadcast ();
1343             }
1344             NS_LOG_DEBUG ("Send_RREQ_with_id" << headerEXPL.GetId () << "to_socket");
1345             m_lastBcastTime = Simulator::Now ();
1346             //SendTo (socket, packet, Ipv4Address ("255.255.255.255"), headerEXPL.GetSerializedSize ());
1347             Simulator::Schedule (Time (Milliseconds (m_uniformRandomVariable->GetInteger (0, 10))),
1348                                 &antcommProtocol::SendTo, this, socket, packet, destination);
1349         }
1350     }
1351 }
1352
1353 void antcommProtocol::SendREPA (antcommEXPLHeader const & headerEXPL, antcommRoutingTableEntry
1354     const & toOrigin)
1355 {
1356     NS_LOG_FUNCTION (this << toOrigin.GetDestination ());
1357     NS_LOG_INFO ("This is a function sending reply packets");
1358     std::cout << "The_dst_from_rreq_packet_is" << headerEXPL.GetDst() << std::endl;

```

```

1355     std::cout << "The origin I read from to origin routing table entry is" << toOrigin.GetDestination
1356         () << std::endl;
1357     antcommREPAHeader headerRepa (0, /*Time to Live*/255, toOrigin.GetDestination (), /*myNodeID */
1358         headerEXPL.GetDst ());
1359     Ptr < Packet > packet = Create < Packet > ();
1360     packet->AddHeader (headerRepa);
1361     TypeHeader tHeader (REPA);
1362     packet->AddHeader (tHeader);
1363     Ptr < Socket > socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
1364     NS_ASSERT (socket);
1365     std::cout << "Socket is confirmed" << std::endl;
1366     socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), antcomm_PORT));
1367     //SendTo (socket, packet, Ipv4Address ("255.255.255.255"), headerRepa.GetSerializedSize ());
1368     //}
1369 }
1370
1371 void antcommProtocol::SendREREPA ()
1372 {
1373     NS_LOG_FUNCTION (this);
1374
1375     for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.
1376         begin (); j != m_socketAddresses.end (); ++j)
1377     {
1378         Ptr < Socket > socket = j->first;
1379         //uint16_t myNodeID = m_ipv4->GetObject<Node> ()->GetId (); //testing by commenting out
1380         antcommREREPAHeader header (header.GetHopCount (), 255, header.GetPheromone (), 0, header.
1381             GetOrigin (), header.GetDst ()); // the same
1382         Ptr < Packet > packet = Create < Packet > ();
1383         packet->AddHeader (header);
1384         TypeHeader tHeader (RE_REPA);
1385         packet->AddHeader (tHeader);
1386         //SendTo (this, socket, packet, Ipv4Address ("255.255.255.255"));
1387     }
1388 }
1389
1390 void antcommProtocol::SendHello ()
1391 {
1392     NS_LOG_FUNCTION (this);
1393     NS_LOG_DEBUG ("Hello message sent");
1394     Ptr < Node > node = m_ipv4->GetObject< Node > ();
1395     Ptr<EnergySourceContainer> EnergySourceContainerOnNode = node->GetObject<EnergySourceContainer>
1396         ();
1397     // Convert the type from EnergySourceContainer to BasicEnergy Source by using Dynamic Cast
1398     Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (
1399         EnergySourceContainerOnNode->Get(0));
1400
1401     NS_LOG_DEBUG ("Remaining energy in send hello is" << basicSourcePtr->GetRemainingEnergy ());
1402     if (basicSourcePtr->GetRemainingEnergy () == 0 || basicSourcePtr->GetRemainingEnergy () < 0)
1403     {
1404         //Get the radio model from BasicEnergySource installed on node
1405         Ptr<DeviceEnergyModel> basicRadioModelPtr = basicSourcePtr->FindDeviceEnergyModels ("ns3::
1406             WifiRadioEnergyModel").Get(0);
1407         // Set all the currents to zero such that the dead node doesn't consume energy
1408         basicRadioModelPtr->SetAttribute ("RxCurrentA", DoubleValue (0.0));
1409         basicRadioModelPtr->SetAttribute ("TxCurrentA", DoubleValue (0.0));
1410         basicRadioModelPtr->SetAttribute ("IdleCurrentA", DoubleValue (0.0));
1411         basicRadioModelPtr->SetAttribute ("CcaBusyCurrentA", DoubleValue (0.0));
1412         NS_LOG_DEBUG ("Shutting down the interface at time" << Simulator::Now ().GetSeconds ());
1413         std::cout << "Shutting down the interface at time" << Simulator::Now ().GetSeconds () << std
1414             ::endl;
1415         // 1 in the SetDown parameter is the Loopback interface index
1416         m_ipv4->SetDown (1);
1417     }
1418 }
1419
1420 for (std::map < Ptr < Socket >, Ipv4InterfaceAddress >::const_iterator j = m_socketAddresses.
1421     begin (); j != m_socketAddresses.end (); ++j)
1422 {
1423     Ptr < Socket > socket = j->first;
1424     Ipv4InterfaceAddress iface = j->second;
1425     antcommHelloHeader helloHeader (1, 0, basicSourcePtr->GetRemainingEnergy (), iface.GetLocal ()
1426         , iface.GetLocal (), Time (AllowedHelloLoss * HelloInterval)); // for source ip, it is
1427         iface.getlocal ()
1428     Ptr < Packet > packet = Create < Packet > ();
1429     packet->AddHeader (helloHeader);
1430     TypeHeader tHeader (HELLO);
1431     packet->AddHeader (tHeader);
1432     Ipv4Address destination;

```



```

1424         if (iface.GetMask () == Ipv4Mask::GetOnes ())
1425         {
1426             destination = Ipv4Address ("255.255.255.255");
1427         }
1428         else
1429         {
1430             destination = iface.GetBroadcast ();
1431         }
1432         Simulator::Schedule ( Time (Milliseconds (m_uniformRandomVariable->GetInteger (0, 100))),
1433                               &antcommProtocol::SendTo, this, socket, packet, destination);
1434     }
1435 }
1436
1437 }
1438
1439 void antcommProtocol::HelloTimerExpire ()
1440 {
1441     NS_LOG_FUNCTION (this);
1442     Time offset = Time (Seconds (0));
1443     if (m_lastBcastTime > Time (Seconds (0)))
1444     {
1445         offset = Simulator::Now () - m_lastBcastTime;
1446         NS_LOG_DEBUG ("Hello_deferred_due_to_last_bcast_at:" << m_lastBcastTime);
1447     }
1448     else
1449     {
1450         SendHello ();
1451     }
1452     m_helloTimer.Cancel ();
1453     Time diff = HelloInterval - offset;
1454     m_helloTimer.Schedule (std::max (Time (Seconds (0)), diff));
1455     //m_helloTimer.Schedule(Seconds (m_helloInterval));
1456     m_lastBcastTime = Time (Seconds (0));
1457 }
1458 void antcommProtocol::ExplRateLimitTimerExpire ()
1459 {
1460     NS_LOG_FUNCTION (this);
1461     m_explCount = 0;
1462     m_forwardTimer.Schedule (Seconds (1));
1463 }
1464
1465 double antcommProtocol::GetPheromone ()
1466 {
1467     Time tm = Simulator::Now ();
1468     Time offset = tm - m_updatedTimeOfPheromone;
1469     double x = 0.0;
1470     x = exp (-offset.GetSeconds () * m_pheromoneDecayRate) * m_pheromone;
1471     m_pheromone = std::max (x, m_epsilon);
1472     m_updatedTimeOfPheromone = tm;
1473     return m_pheromone;
1474 }
1475
1476 void antcommProtocol::DoInitialize (void)
1477 {
1478     NS_LOG_FUNCTION (this);
1479     uint32_t startTime;
1480     m_helloTimer.SetFunction (&antcommProtocol::HelloTimerExpire, this);
1481     startTime = m_uniformRandomVariable->GetInteger (0, 100);
1482     //std::cout << "Starting at time" << startTime << "ms" << std::endl;
1483     NS_LOG_DEBUG ("Starting_at_time" << startTime << "ms");
1484     m_helloTimer.Schedule (Milliseconds (startTime));
1485
1486     Ipv4RoutingProtocol::DoInitialize ();
1487 }
1488
1489 } //namespace antcomm
1490 } //namespace ns3

```

---

---

```

1  /*
2  * Copyright (c) 2015 SKKU Networking Laboratory
3  *
4  * Authors: Soon-gyo Jung <soongyo@skku.edu>
5  */
6  #ifndef antcommPROTOCOL_H
7  #define antcommPROTOCOL_H
8
9  #define INITIAL_HOP_COUNT    9999
10 //50m
11 #define TRANSMISSION_RANGE    60.0
12 #define DISTANCE_FOR_TIMER    0.5
13
14 #define INITIAL_PHEROMONE     1.0
15 #define INCREMENTAL_PHEROMONE 1E-5
16 #define PHEROMONE_DECAY_RATE  0.5
17 #define EPSILON 1.0E-6
18 #define PHEROMONE_THRESHOLD 3*INITIAL_PHEROMONE
19 #define TIME_MIN_FOR_WAITING_TIMER 30
20 #define TIME_MAX_FOR_WAITING_TIMER 300
21 #define ALPHA 0.33
22 #define BETA 0.33
23 #define GAMMA 0.33
24
25 #define TIME_FOR_HELLO_TIMER 4
26 //s
27 #define EXPLORATION_TIME 10
28 #define REPAIR_TIME 900
29
30 #include "antcomm-packet.h"
31 #include "antcomm-rtable.h"
32 #include "antcomm-rqueue.h"
33 #include "antcomm-dpd.h"
34 #include "antcomm-neighbor.h"
35 #include "ns3/node.h"
36 #include "randomx.h"
37 #include "ns3/random-variable-stream.h"
38 #include "ns3/output-stream-wrapper.h"
39 #include "ns3/ipv4-routing-protocol.h"
40 #include "ns3/ipv4-interface.h"
41 #include "ns3/ipv4-l3-protocol.h"
42 #include "ns3/network-module.h"
43 #include <map>
44 #include <iostream>
45
46 namespace ns3
47 {
48     namespace antcomm
49     {
50
51         enum Phase
52         {
53             EXPLORATION = 1, //!< Exploration
54             FORAGING = 2,  //!< Foraging
55             REPAIRING = 3  //!< Repairing
56         };
57
58         enum PheromoneUpdateType
59         {
60             EVENT = 1,
61             RECALCULATION = 2
62         };
63
64         /**
65          * \ingroup antcomm
66          *
67          * \brief antcomm protocol
68          */
69         class antcommProtocol:public Ipv4RoutingProtocol
70         {
71         public:
72
73             static TypeId GetTypeId (void);
74             static const uint32_t antcomm_PORT;
75
76             antcommProtocol ();
77
78             virtual ~ antcommProtocol ();
79             virtual void DoDispose ();
80

```

```

81
82 // Inherited from Ipv4RoutingProtocol
83 Ptr < Ipv4Route > RouteOutput (Ptr < Packet > p, const Ipv4Header & header, Ptr < NetDevice >
    oif, Socket::SocketErrno & sockerr);
84 bool RouteInput (Ptr < const Packet > p, const Ipv4Header & header, Ptr < const NetDevice > idev
    ,
85         UnicastForwardCallback ucb, MulticastForwardCallback mcb,
86         LocalDeliverCallback lcb, ErrorCallback ecb);
87 Ptr < Ipv4Route > LoopbackRoute (const Ipv4Header & hdr, Ptr < NetDevice > oif) const;
88 virtual void NotifyInterfaceUp (uint32_t interface);
89 virtual void NotifyInterfaceDown (uint32_t interface);
90 virtual void NotifyAddAddress (uint32_t interface, Ipv4InterfaceAddress address);
91 virtual void NotifyRemoveAddress (uint32_t interface, Ipv4InterfaceAddress address);
92 virtual void SetIpv4 (Ptr < Ipv4 > ipv4);
93 virtual void PrintRoutingTable (Ptr < OutputStreamWrapper > stream) const;
94
95 Time GetMaxQueueTime () const { return MaxQueueTime; }
96 void SetMaxQueueTime (Time t);
97 uint32_t GetMaxQueueLen () const { return MaxQueueLen; }
98 void SetMaxQueueLen (uint32_t len);
99 void SetBroadcastEnable (bool f) { EnableBroadcast = f; }
100 bool GetBroadcastEnable () const { return EnableBroadcast; }
101
102 int64_t AssignStreams (int64_t stream);
103
104
105 void UpdateCurrentPhase (Phase phase);
106
107 void SendDATAPACKET (bool isFirstPacket, uint16_t count,
108                     uint32_t packetID);
109
110 void PrintInformation (Ptr < OutputStreamWrapper > stream) const;
111
112
113 protected:
114     virtual void DoInitialize (void);
115
116
117 private:
118
119     uint32_t m_numberOfReceivedPacket;
120
121     uint16_t EXPLRateLimit; ///< Maximum number of EXPL per second.
122     double m_averageHopDistanceOfReceivedPackets;
123     double m_averageEndToEndDelays;
124
125     uint16_t m_numberOfReceivedAnts;
126     Time ActiveRouteTimeout;
127
128     uint32_t NetDiameter; ///< Net diameter measures the maximum possible number of hops between two
        nodes in the network
129     /**
130      * NodeTraversalTime is a conservative estimate of the average one hop traversal time for packets
131      * and should include queuing delays, interrupt processing times and transfer times.
132      */
133     Time NodeTraversalTime;
134     Time NetTraversalTime; ///< Estimate of the average net traversal time.
135     Time PathDiscoveryTime; ///< Estimate of maximum time needed to find route in network.
136
137     //Phase
138     Phase m_currentPhase;
139
140     Time HelloInterval; // just now added
141
142     uint32_t AllowedHelloLoss;
143
144     uint32_t m_explorationTime;
145     ///< Exprolation timer
146     Timer m_explorationTimer;
147     ///< Repair timer
148     uint32_t m_repairTime;
149     Timer m_repairTimer;
150
151
152     double m_transmissionRange;
153     double m_distanceForTimer;
154     bool EnableBroadcast;
155     ///< IP protocol
156     Ptr < Ipv4 > m_ipv4;
157     ///< Loopback device used to defer
158     Ptr < NetDevice > m_lo;

```

```

159    /// Raw unicast socket per each IP interface, map socket -> iface address (IP + mask)
160    std::map < Ptr < Socket >, Ipv4InterfaceAddress > m_socketAddresses;
161    /// Raw subnet directed broadcast socket per each IP interface, map socket -> iface address (IP +
162    mask)
162    std::map < Ptr < Socket >, Ipv4InterfaceAddress > m_socketSubnetBroadcastAddresses;
163
164
165    ///Pheromone
166    double m_pheromone;
167
168    ///Initial pheromone
169    double m_initialPheromone;
170
171    ///Pheromone decay rate
172    float m_pheromoneDecayRate;
173    ///Time of pheromone updated
174    Time m_updatedTimeOfPheromone;
175    double m_epsilon;
176
177    ///IP4 Address of sink
178    ///Ipv4Address m_sink;
179
180    ///Minimum time (ms) for waiting timer
181    uint32_t m_timeMinForWaitingTimer;
182    ///Maximum time (ms) for waiting timer
183    uint32_t m_timeMaxForWaitingTimer;
184    ///Weight parameters for selection probability
185    double m_alpha;
186    double m_beta;
187    double m_gamma;
188
189    /// Routing table
190    RoutingTable m_routingTable;
191    /// Used by the routing layer to buffer packets to which it does not have a route.
192    uint32_t MaxQueueLen; ///< The maximum number of packets that we allow a routing protocol to
193    buffer.
194
195    Time MaxQueueTime; ///< The maximum period of time that a routing protocol is allowed to buffer a
196    packet for.
197
198    RequestQueue m_queue;
199
200    /// Hello Interval
201    uint32_t m_helloInterval;
202
203    /// Broadcast ID
204    uint32_t m_requestId;
205    /// Hello timer
206    Timer m_helloTimer;
207    /// Handle duplicated RREQ
208    IdCache m_rreqIdCache;
209    /// Handle duplicated broadcast/multicast packets
210    DuplicatePacketDetection m_dpd;
211
212    /// Number of EXPLs used for EXPL rate control
213    uint16_t m_explCount;
214
215    Neighbors m_neighborTable;
216
217    /// Keep track of the last bcast time
218    Time m_lastBcastTime;
219
220    Timer m_forwardTimer;
221
222    Timer m_backwardTimer;
223
224    Timer m_retryTimer;
225
226    uint32_t numberOfDeferredPacket;
227
228    /// Waiting timer
229    Timer m_waitingTimer;
230
231
232    /// Provides uniform random variables.
233    Ptr < UniformRandomVariable > m_uniformRandomVariable;
234
235
236    private:

```

```

237
238     /// Start protocol operation
239     void Start ();
240
241     bool UpdateRouteLifeTime (Ipv4Address addr, Time lt);
242
243     /// Check that packet is send from own interface
244     bool IsMyOwnAddress (Ipv4Address src);
245
246
247     /// Receive and process control packet
248     void Recvantcomm (Ptr < Socket > socket);
249     /// Receive HELLO
250     void RecvHELLO (Ptr<Packet> p, Ipv4Address my, Ipv4Address sender);
251     /// Make the routing table after hello messages have been exchanged
252     /// Ipv4Address origin is originator of foraging ant
253     /// Ipv4Address my is Ipv4Address of myself
254     /// Ipv4Address src is Ipv4Address of my neighbor
255     void MakeRoutingTable (Ipv4Address origin, Ipv4Address my, Ipv4Address src);
256     /// Receive EXPL
257     void RecvEXPL (Ptr<Packet> p, Ipv4Address my, Ipv4Address src);
258
259     /// Receive REPA
260     void RecvREPA (Ptr<Packet> p, Ipv4Address my, Ipv4Address sender);
261     /** Send RREP by intermediate node
262     * \param toDst routing table entry to destination
263     * \param toOrigin routing table entry to originator
264     */
265     void SendReplyByIntermediateNode (antcommRoutingTableEntry & toDst, antcommRoutingTableEntry &
        toOrigin);
266
267     bool Forwarding (Ptr<const Packet> p, const Ipv4Header & header, UnicastForwardCallback ucb,
        ErrorCallback ecb);
268     /// Queue packet and send route requeste
269     void DeferredRouteOutput (Ptr < const Packet > p, const Ipv4Header & header,
        UnicastForwardCallback ucb, ErrorCallback ecb);
270
271     /// Find unicast socket with local interface address iface
272     Ptr < Socket > FindSocketWithInterfaceAddress (Ipv4InterfaceAddress iface) const;
273     /// Find subnet directed broadcast socket with local interface address iface
274     Ptr < Socket > FindSubnetBroadcastSocketWithInterfaceAddress (Ipv4InterfaceAddress iface)
        const;
275
276     /// For antcomm's packet sending
277     void SendPacketFromQueue (Ipv4Address dst, Ptr < Ipv4Route > route);
278
279     void SendHello ();
280
281     void SendEXPL (Ipv4Address dst);
282
283     void SendREPA (antcommEXPLHeader const & EXPLHeader, antcommRoutingTableEntry const & toOrigin);
284     void SendREREPA ();
285     void SendTo (Ptr < Socket > socket, Ptr < Packet > packet, Ipv4Address destination);
286     /// Schedule next send of hello message
287     void HelloTimerExpire ();
288
289     /// Reset EXPL count and schedule EXPL rate limit timer with delay 1 sec.
290     void ExplRateLimitTimerExpire ();
291
292     double GetPheromone();
293
294 };
295
296 }
297
298 #endif     /* antcommPROTOCOL_H */

```

---

---

```

1  /*
2  * Copyright (c) 2015 SKKU Networking Laboratory
3  *
4  * Authors: Soon-gyo Jung <soongyo@skku.edu>
5  */
6  #include "antcomm-rqueue.h"
7  #include <algorithm>
8  #include <functional>
9  #include "ns3/ipv4-route.h"
10 #include "ns3/socket.h"
11 #include "ns3/log.h"
12
13 namespace ns3
14 {
15
16     NS_LOG_COMPONENT_DEFINE ("antcommRequestQueue");
17
18     namespace antcomm
19     {
20
21         uint32_t RequestQueue::GetSize ()
22         {
23             Purge ();
24             return m_queue.size ();
25         }
26
27         bool RequestQueue::Enqueue (QueueEntry & entry)
28         {
29             Purge ();
30             for (std::vector < QueueEntry >::const_iterator i = m_queue.begin ();
31                  i != m_queue.end (); ++i)
32             {
33                 if ((i->GetPacket ()->GetUid () == entry.GetPacket ()->GetUid ())
34                     && (i->GetIpv4Header ().GetDestination () ==
35                        entry.GetIpv4Header ().GetDestination ()))
36                     return false;
37             }
38             entry.SetExpireTime (m_queueTimeout);
39             if (m_queue.size () == m_maxLen)
40             {
41                 Drop (m_queue.front (), "Drop the most aged packet"); // Drop the most aged packet
42                 m_queue.erase (m_queue.begin ());
43             }
44             m_queue.push_back (entry);
45             return true;
46         }
47
48         void RequestQueue::DropPacketWithDst (Ipv4Address dst)
49         {
50             NS_LOG_FUNCTION (this << dst);
51             Purge ();
52             for (std::vector < QueueEntry >::iterator i = m_queue.begin ();
53                  i != m_queue.end (); ++i)
54             {
55                 if (IsEqual (*i, dst))
56                 {
57                     Drop (*i, "DropPacketWithDst");
58                 }
59             }
60             m_queue.erase (std::remove_if (m_queue.begin (), m_queue.end (),
61                                             std::bind2nd (std::ptr_fun (RequestQueue::
62                                             IsEqual), dst)),
63                           m_queue.end ());
64         }
65     }
66
67     bool RequestQueue::Dequeue (Ipv4Address dst, QueueEntry & entry)
68     {
69         Purge ();
70         for (std::vector < QueueEntry >::iterator i = m_queue.begin ();
71              i != m_queue.end (); ++i)
72         {
73             if (i->GetIpv4Header ().GetDestination () == dst)
74             {
75                 entry = *i;
76                 m_queue.erase (i);
77                 return true;
78             }
79         }
80         return false;

```

```

81     }
82
83     bool RequestQueue::Find (Ipv4Address dst)
84     {
85         for (std::vector < QueueEntry >::const_iterator i = m_queue.begin ();
86              i != m_queue.end (); ++i)
87             {
88                 if (i->GetIpv4Header ().GetDestination () == dst)
89                     return true;
90             }
91         return false;
92     }
93
94     struct IsExpired
95     {
96         bool operator () (QueueEntry const &e) const
97         {
98             return (e.GetExpireTime () < Seconds (0));
99         }
100     };
101
102     void RequestQueue::Purge ()
103     {
104         IsExpired pred;
105         for (std::vector < QueueEntry >::iterator i = m_queue.begin ();
106              i != m_queue.end (); ++i)
107             {
108                 if (pred (*i))
109                 {
110                     Drop (*i, "Drop_outdated_packet");
111                 }
112             }
113         m_queue.erase (std::remove_if (m_queue.begin (), m_queue.end (), pred),
114                        m_queue.end ());
115     }
116
117     void RequestQueue::Drop (QueueEntry en, std::string reason)
118     {
119         NS_LOG_LOGIC (reason << en.GetPacket ()->GetUid () << " " << en.
120                      GetIpv4Header ().GetDestination ());
121         en.GetErrorCallback ()(en.GetPacket (), en.GetIpv4Header (),
122                               Socket::ERROR_NOROUTETOHOST);
123         return;
124     }
125 }
126 }
127 }

```

---

---

```

1  /*
2  * Copyright (c) 2015 SKKU Networking Laboratory
3  *
4  * Authors: Soon-gyo Jung <soongyo@skku.edu>
5  */
6  #ifndef antcomm_RQUEUE_H
7  #define antcomm_RQUEUE_H
8
9  #include <vector>
10 #include "ns3/ipv4-routing-protocol.h"
11 #include "ns3/simulator.h"
12
13
14 namespace ns3 {
15 namespace antcomm {
16
17 /**
18  * \ingroup antcomm
19  * \brief antcomm Queue Entry
20  */
21 class QueueEntry {
22 public:
23     typedef Ipv4RoutingProtocol::UnicastForwardCallback UnicastForwardCallback;
24     typedef Ipv4RoutingProtocol::ErrorCallback ErrorCallback;
25     /// c-tor
26     QueueEntry (Ptr<const Packet> pa = 0, Ipv4Header const & h = Ipv4Header (),
27                 UnicastForwardCallback ucb = UnicastForwardCallback (),
28                 ErrorCallback ecb = ErrorCallback (), Time exp = Simulator::Now ()) :
29         m_packet (pa), m_header (h), m_ucb (ucb), m_ecb (ecb),
30         m_expire (exp + Simulator::Now ()) {
31     }
32
33     /**
34      * Compare queue entries
35      * \return true if equal
36      */
37     bool operator== (QueueEntry const & o) const {
38         return ((m_packet == o.m_packet) && (m_header.GetDestination () == o.m_header.
39             GetDestination ()) && (m_expire == o.m_expire));
40     }
41
42     /// Fields
43     UnicastForwardCallback GetUnicastForwardCallback () const { return m_ucb; }
44     void SetUnicastForwardCallback (UnicastForwardCallback ucb) { m_ucb = ucb; }
45     ErrorCallback GetErrorCallback () const { return m_ecb; }
46     void SetErrorCallback (ErrorCallback ecb) { m_ecb = ecb; }
47     Ptr<const Packet> GetPacket () const { return m_packet; }
48     void SetPacket (Ptr<const Packet> p) { m_packet = p; }
49     Ipv4Header GetIpv4Header () const { return m_header; }
50     void SetIpv4Header (Ipv4Header h) { m_header = h; }
51     void SetExpireTime (Time exp) { m_expire = exp + Simulator::Now (); }
52     Time GetExpireTime () const { return m_expire - Simulator::Now (); }
53
54 private:
55     /// Data packet
56     Ptr<const Packet> m_packet;
57     /// IP header
58     Ipv4Header m_header;
59     /// Unicast forward callback
60     UnicastForwardCallback m_ucb;
61     /// Error callback
62     ErrorCallback m_ecb;
63     /// Expire time for queue entry
64     Time m_expire;
65 };
66 /**
67  * \ingroup aodv
68  * \brief antcomm route request queue
69  *
70  * Since antcomm is an on demand routing we queue requests while looking for route.
71  */
72 class RequestQueue {
73 public:
74     /// Default c-tor
75     RequestQueue (uint32_t maxLen, Time routeToQueueTimeout) :
76         m_maxLen (maxLen), m_queueTimeout (routeToQueueTimeout) {
77     }
78     /// Push entry in queue, if there is no entry with the same packet and destination address
79     in queue.

```



```

79         bool Enqueue (QueueEntry & entry);
80         /// Return first found (the earliest) entry for given destination
81         bool Dequeue (Ipv4Address dst, QueueEntry & entry);
82         /// Remove all packets with destination IP address dst
83         void DropPacketWithDst (Ipv4Address dst);
84         /// Finds whether a packet with destination dst exists in the queue
85         bool Find (Ipv4Address dst);
86         /// Number of entries
87         uint32_t GetSize ();
88
89         // Fields
90         uint32_t GetMaxQueueLen () const { return m_maxLen; }
91         void SetMaxQueueLen (uint32_t len) { m_maxLen = len; }
92         Time GetQueueTimeout () const { return m_queueTimeout; }
93         void SetQueueTimeout (Time t) { m_queueTimeout = t; }
94
95     private:
96
97         std::vector<QueueEntry> m_queue;
98         /// Remove all expired entries
99         void Purge ();
100        /// Notify that packet is dropped from queue by timeout
101        void Drop (QueueEntry en, std::string reason);
102        /// The maximum number of packets that we allow a routing protocol to buffer.
103        uint32_t m_maxLen;
104        /// The maximum period of time that a routing protocol is allowed to buffer a packet for,
105        seconds.
106        Time m_queueTimeout;
107        static bool IsEqual (QueueEntry en, const Ipv4Address dst) { return (en.GetIpv4Header ().
108            GetDestination () == dst); }
109    };
110 }
111 }
112
113 #endif /* antcomm_RQUEUE_H */

```

---

---

```

1  /*
2  * Copyright (c) 2015 SKKU Networking Laboratory
3  *
4  * Authors: Soon-gyo Jung <soongyo@skku.edu>
5  */
6
7  #include "antcomm-rtable.h"
8  #include "antcomm-helper.h"
9  #include <algorithm>
10 #include <math.h>
11 #include <iomanip>
12 #include "ns3/simulator.h"
13 #include "ns3/random-variable-stream.h"
14 #include "ns3/log.h"
15 #define MAX 100
16
17 namespace ns3
18 {
19
20     NS_LOG_COMPONENT_DEFINE ("antcommRoutingTable");
21
22     namespace antcomm
23     {
24
25         /*
26          * The antcomm Information Table
27          */
28         antcommRoutingTableEntry::antcommRoutingTableEntry (Ptr<NetDevice> dev, Ipv4Address dst, uint16_t
            hopCount, double pheromone, double residualEnergy, double rssi, Ipv4Address nextHop,
29             Ipv4InterfaceAddress iface, Time updatedTime):
            m_hopCount (hopCount), m_pheromone (pheromone), m_residualEnergy (residualEnergy), m_rssi (rssi)
            , m_iface (iface), m_updatedTime (updatedTime + Simulator::Now ())
30         {
31             m_ipv4Route = Create<Ipv4Route> ();
32             m_ipv4Route->SetDestination (dst);
33             m_ipv4Route->SetGateway (nextHop);
34             m_ipv4Route->SetSource (m_iface.GetLocal ());
35             m_ipv4Route->SetOutputDevice (dev);
36         }
37
38         antcommRoutingTableEntry::~antcommRoutingTableEntry ()
39         {
40         }
41
42         bool antcommRoutingTableEntry::InsertPrecursor (Ipv4Address id)
43         {
44             NS_LOG_FUNCTION (this << id);
45             if (LookupPrecursor (id))
46             {
47                 return false;
48             }
49             else
50             {
51                 m_precursorList.push_back (id);
52                 return true;
53             }
54         }
55
56         bool antcommRoutingTableEntry::LookupPrecursor (Ipv4Address id)
57         {
58             NS_LOG_FUNCTION (this << id);
59             for (std::vector<Ipv4Address>::const_iterator i = m_precursorList.begin (); i
60                 != m_precursorList.end (); ++i)
61             {
62                 if (*i == id)
63                 {
64                     NS_LOG_LOGIC ("Precursor_" << id << "_found");
65                     return true;
66                 }
67             }
68             NS_LOG_LOGIC ("Precursor_" << id << "_not_found");
69             return false;
70         }
71
72         void antcommRoutingTableEntry::Print (Ptr < OutputStreamWrapper > stream) const
73         {
74             *stream->GetStream () << m_ipv4Route->GetGateway ()/*m_dest*/ << "\t"
75             << m_hopCount << "\t" << m_pheromone << "\t" << m_rssi << "\t"
76             << m_residualEnergy << "\t" << std::setw (14) << m_updatedTime.GetSeconds () << "\n";
77         }
78
79         /*

```

```

78      The Routing Table
79      */
80      bool RoutingTable::LookupRoute (Ipv4Address dst, Ipv4Address neighbor, antcommRoutingTableEntry &
      nt)
81      {
82
83          NS_LOG_FUNCTION (this << dst << neighbor);
84          RTable::const_iterator it;
85          Neighbor::const_iterator jt;
86          if(m_routes.empty())
87          {
88              NS_LOG_LOGIC("Backward_learning_Route_to" << dst << "not_found;m_routes_is_empty");
89              return false;
90          }
91          it = m_routes.find (dst);
92          if (it != m_routes.end ()) {
93              pout << "Backward_learning_Destination_Exists:" << it->first << std::endl;
94              jt = it->second.find (neighbor);
95              if (jt != it->second.end ())
96              {
97                  nt = m_routes[it->first][jt->first];
98                  NS_LOG_LOGIC ("Backward_learning_Route_to" << dst << "found");
99                  pout << "Backward_Neighbor_Exists:" << jt->first << std::endl;
100                  return true;
101              }
102          }
103          NS_LOG_LOGIC ("Backward_learning_Route_to" << dst << "not_found");
104          return false;
105      }
106
107      bool RoutingTable::LookupRoute2 (Ipv4Address dst, antcommRoutingTableEntry & nt)
108      {
109          /* Rules for finding the good neighbor:
110          a) find the destination node
111          b) If the iterator is pointing towards end then it means destination not found
112          c) Otherwise scan neighbor
113          d) Use probabilistic rule to select the good neighbor
114          e) Return its entry and set the parameter, "nt" to the entry you found above */
115          NS_LOG_FUNCTION(this << dst);
116          RTable::const_iterator it;
117          Neighbor::const_iterator jt;
118          antcommRoutingTableEntry entry[MAX];
119          double probability[MAX];
120          int i;
121          double cum = 0;
122          int counter = 0;
123          double sum = 0;
124          double pvt;
125          if(m_routes.empty())
126          {
127              NS_LOG_LOGIC("Route_to" << dst << "not_found;m_routes_is_empty");
128              return false;
129          }
130          it = m_routes.find(dst);
131          if(it == m_routes.end())
132          {
133              NS_LOG_LOGIC("Route_to" << dst << "not_found");
134              return false;
135          }
136          if(it->second.begin() == it->second.end())
137          {
138              NS_LOG_LOGIC("There_are_no_neighbors_present,hence_cant_forward_the_packet");
139              return false;
140          }
141          for (jt = it->second.begin(); jt != it->second.end(); ++jt)
142          {
143              probability[counter] = pow((jt->second).GetPheromone(), 1)*pow((jt->second).GetHopCount(),
              3.2)*
144              pow((jt->second).GetResidualEnergy(), 1.8)*pow((jt->second).GetRssi(), 2.3); //2.8 2.3 is
              current
145              entry[counter] = jt->second;
146              sum += probability[counter];
147              counter++;
148          }
149          NS_LOG_DEBUG ( "Neighbor_counter_is" << counter);
150          NS_LOG_INFO ( "Number_of_Neighbors_are" << counter );
151          Ptr<UniformRandomVariable> x = CreateObject<UniformRandomVariable>();
152          pvt = x->GetValue(0.0, sum);
153          // for (cum = probability[i=0]; i < counter && pvt >= cum; cum += probability[++i]) // changed ++ here
154          // ;

```

```

155     for (i=0; i < counter; ++i)
156     {
157         cum += probability[i];
158         if (pvt <= cum)
159             break;
160     }
161
162     NS_ASSERT(i != counter);
163
164     //std::cout << i<<"th Node ID is selected as neighbor"<< std::endl;
165
166     nt = entry[i];
167     NS_LOG_LOGIC ("Route_to_" << dst << "_found");
168     /*jt = it->second.begin ();
169
170     while (i-- > 0)
171         jt++;
172
173     nt = jt->second;*/
174     //nt = m_routes[it->first][jt->first];
175     return true;
176 }
177
178 bool RoutingTable::DeleteRoute (Ipv4Address dst)
179 {
180     NS_LOG_FUNCTION (this << dst );
181     if (m_routes.erase (dst) != 0)
182     {
183         NS_LOG_LOGIC ("Route_deletion_of_" << dst << "_successful");
184         return true;
185     }
186     NS_LOG_LOGIC ("Route_deletion_of_" << dst << "_not_successful");
187     return false;
188 }
189
190
191 bool RoutingTable::AddRoute (antcommRoutingTableEntry & it)
192 {
193     NS_LOG_FUNCTION (this);
194     //m_routes [it.GetDestination ()].insert (std::make_pair (it.GetNextHop (), it));
195     //pout << "Adding Route ( " <<it.GetDestination() << " , " << it.GetNextHop() <<" )\n";
196     m_routes[it.GetDestination()][it.GetNextHop()] = it;
197     return true;
198 }
199
200
201 bool RoutingTable::Update (antcommRoutingTableEntry & nt)
202 {
203     NS_LOG_FUNCTION (this);
204     RTable::iterator it;
205     Neighbor::iterator jt;
206     it = m_routes.find (nt.GetDestination());
207     if (it == m_routes.end ())
208     {
209         return false;
210     }
211     jt = it->second.find(nt.GetNextHop());
212     m_routes[it->first][jt->first] = nt; //jt->second = nt;
213     return true;
214 }
215
216
217 Ipv4Address RoutingTable::NextHopforDst (antcommRoutingTableEntry & nt)
218 {
219     RTable::const_iterator it;
220     Neighbor::const_iterator jt;
221     double probability[MAX];
222     Ipv4Address none = "0.0.0.0";
223     int counter = 0;
224     int i;
225     double sum = 0;
226     double cum = 0;
227     double pvt;
228     if(m_routes.empty())
229     {
230         NS_LOG_LOGIC("Route_to" << nt.GetDestination () << "not_found;_m_routes_is_empty");
231         return none;
232     }
233     it = m_routes.find(nt.GetDestination ());
234     if(it == m_routes.end())
235     {

```

```

236     NS_LOG_LOGIC("Route_to_" << nt.GetDestination () << "not_found");
237     return none;
238 }
239
240 for (jt = it->second.begin(); jt != it->second.end(); ++jt)
241 {
242     probability[counter] = pow((jt->second).GetPheromone(), 1) * pow((jt->second).GetHopCount(),
243         2);
244     sum += probability[counter];
245     counter++;
246 }
247 Ptr<UniformRandomVariable> x = CreateObject<UniformRandomVariable> ();
248 pvt = x->GetValue(0.0, sum);
249 for (cum = probability[i=0]; i < counter && pvt>=cum; cum+=probability[++i])
250 ;
251 jt = it->second.begin();
252 while (i-- > 0)
253 {
254     jt++;
255 }
256 return jt->first;
257 }
258
259 void RoutingTable::Print (Ptr < OutputStreamWrapper > stream) const
260 {
261
262     RTable::const_iterator ot;
263     Neighbor::const_iterator it;
264     *stream->GetStream () << "\nantcomm_information_table\n"
265     << "ID\tHopCount\tPheromone\tRSSI\tResidualEnergy\tUpdated_Time\n";
266
267     /*std::cout << "map size= "<< m_routes.size() << std::endl;
268     ot = m_routes.begin();
269     std::cout << "map2 first size= " <<(ot->second).size() << std::endl;
270     ot = m_routes.end(); ot--;
271     std::cout << "map2 last size= " <<(ot->second).size() << std::endl;*/
272     for (ot = m_routes.begin (); ot != m_routes.end (); ++ot)
273     {
274         //std::cout << "Printing the destinations " << ot->first << std::endl;
275         for (it = ot->second.begin (); it != ot->second.end (); ++it)
276
277
278         {
279             it->second.Print (stream);
280         }
281     }
282     *stream->GetStream () << "\n";
283
284 }
285
286 }
287 }

```

---

---

```

1  /*
2  * Copyright (c) 2015 SKKU Networking Laboratory, IUPUI
3  *
4  * Authors: Soon-gyo Jung <soongyo@skku.edu>
5  */
6  #ifndef antcomm_RTABLE_H
7  #define antcomm_RTABLE_H
8
9  #define ARUSH_DEBUG_COUT //comment this if you want to suppress the cout-s
10
11 #ifdef ARUSH_DEBUG_COUT
12 #define pout std::cout
13 #else
14 #define pout if(0) std::cout
15 #endif
16
17 #include <stdint.h>
18 #include <cassert>
19 #include <sys/types.h>
20 #include <vector>
21 #include "ns3/ipv4.h"
22 #include "ns3/ipv4-route.h"
23 #include "ns3/timer.h"
24 #include "ns3/net-device.h"
25 #include "ns3/wifi-mac-header.h"
26 #include "ns3/output-stream-wrapper.h"
27
28 namespace ns3
29 {
30
31     namespace antcomm
32     {
33
34         /**
35          * \ingroup antcomm
36          * \brief The table entry used by antcomm protocol
37          */
38
39         class antcommRoutingTableEntry
40         {
41         public:
42
43             antcommRoutingTableEntry (Ptr<NetDevice> dev = 0, Ipv4Address dst = Ipv4Address (),
44                                     uint16_t hopCount = 1, double pheromone = 0.0, double residualEnergy =
45                                     0.0, double rssi = 0.0, Ipv4Address nextHop = Ipv4Address (),
46                                     Ipv4InterfaceAddress iface = Ipv4InterfaceAddress (), Time
47                                     updateTime = Simulator::Now ());
48
49             ~antcommRoutingTableEntry ();
50
51             /**
52              * Insert precursor in precursor list if it doesn't yet exist in the list
53              * \param id precursor address
54              * \return true on success
55              */
56             bool InsertPrecursor (Ipv4Address id);
57
58             /**
59              * Lookup precursor by address
60              * \param id precursor address
61              * \return true on success
62              */
63             bool LookupPrecursor (Ipv4Address id);
64
65             Ipv4Address GetDestination () const
66             {
67                 return m_ipv4Route->GetDestination ();
68             }
69
70             Ptr<Ipv4Route> GetRoute () const
71             {
72                 return m_ipv4Route;
73             }
74
75             void SetRoute (Ptr<Ipv4Route> r)
76             {
77                 m_ipv4Route = r;
78             }
79
80             void SetHopCount (uint16_t hopCount)
81             {
82                 m_hopCount = hopCount;
83             }
84
85             void SetPheromone (double pheromone)
86             {
87                 m_pheromone = pheromone;
88             }
89
90             void SetResidualEnergy (double residualEnergy)
91             {
92                 m_residualEnergy = residualEnergy;
93             }
94
95             void SetRssi (double rssi)
96             {
97                 m_rssi = rssi;
98             }
99
100             void SetNextHop (Ipv4Address nextHop)
101             {
102                 m_nextHop = nextHop;
103             }
104
105             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
106             {
107                 m_iface = iface;
108             }
109
110             void SetUpdateTime (Time updateTime)
111             {
112                 m_updateTime = updateTime;
113             }
114
115             Ptr<NetDevice> GetDevice () const
116             {
117                 return m_dev;
118             }
119
120             Ipv4Address GetDst () const
121             {
122                 return m_dst;
123             }
124
125             uint16_t GetHopCount () const
126             {
127                 return m_hopCount;
128             }
129
130             double GetPheromone () const
131             {
132                 return m_pheromone;
133             }
134
135             double GetResidualEnergy () const
136             {
137                 return m_residualEnergy;
138             }
139
140             double GetRssi () const
141             {
142                 return m_rssi;
143             }
144
145             Ipv4Address GetNextHop () const
146             {
147                 return m_nextHop;
148             }
149
150             Ipv4InterfaceAddress GetInterfaceAddress () const
151             {
152                 return m_iface;
153             }
154
155             Time GetUpdateTime () const
156             {
157                 return m_updateTime;
158             }
159
160             void SetDevice (Ptr<NetDevice> dev)
161             {
162                 m_dev = dev;
163             }
164
165             void SetDst (Ipv4Address dst)
166             {
167                 m_dst = dst;
168             }
169
170             void SetHopCount (uint16_t hopCount)
171             {
172                 m_hopCount = hopCount;
173             }
174
175             void SetPheromone (double pheromone)
176             {
177                 m_pheromone = pheromone;
178             }
179
180             void SetResidualEnergy (double residualEnergy)
181             {
182                 m_residualEnergy = residualEnergy;
183             }
184
185             void SetRssi (double rssi)
186             {
187                 m_rssi = rssi;
188             }
189
190             void SetNextHop (Ipv4Address nextHop)
191             {
192                 m_nextHop = nextHop;
193             }
194
195             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
196             {
197                 m_iface = iface;
198             }
199
200             void SetUpdateTime (Time updateTime)
201             {
202                 m_updateTime = updateTime;
203             }
204
205             void SetNextHop (Ipv4Address nextHop)
206             {
207                 m_nextHop = nextHop;
208             }
209
210             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
211             {
212                 m_iface = iface;
213             }
214
215             void SetUpdateTime (Time updateTime)
216             {
217                 m_updateTime = updateTime;
218             }
219
220             void SetNextHop (Ipv4Address nextHop)
221             {
222                 m_nextHop = nextHop;
223             }
224
225             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
226             {
227                 m_iface = iface;
228             }
229
230             void SetUpdateTime (Time updateTime)
231             {
232                 m_updateTime = updateTime;
233             }
234
235             void SetNextHop (Ipv4Address nextHop)
236             {
237                 m_nextHop = nextHop;
238             }
239
240             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
241             {
242                 m_iface = iface;
243             }
244
245             void SetUpdateTime (Time updateTime)
246             {
247                 m_updateTime = updateTime;
248             }
249
250             void SetNextHop (Ipv4Address nextHop)
251             {
252                 m_nextHop = nextHop;
253             }
254
255             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
256             {
257                 m_iface = iface;
258             }
259
260             void SetUpdateTime (Time updateTime)
261             {
262                 m_updateTime = updateTime;
263             }
264
265             void SetNextHop (Ipv4Address nextHop)
266             {
267                 m_nextHop = nextHop;
268             }
269
270             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
271             {
272                 m_iface = iface;
273             }
274
275             void SetUpdateTime (Time updateTime)
276             {
277                 m_updateTime = updateTime;
278             }
279
280             void SetNextHop (Ipv4Address nextHop)
281             {
282                 m_nextHop = nextHop;
283             }
284
285             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
286             {
287                 m_iface = iface;
288             }
289
290             void SetUpdateTime (Time updateTime)
291             {
292                 m_updateTime = updateTime;
293             }
294
295             void SetNextHop (Ipv4Address nextHop)
296             {
297                 m_nextHop = nextHop;
298             }
299
300             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
301             {
302                 m_iface = iface;
303             }
304
305             void SetUpdateTime (Time updateTime)
306             {
307                 m_updateTime = updateTime;
308             }
309
310             void SetNextHop (Ipv4Address nextHop)
311             {
312                 m_nextHop = nextHop;
313             }
314
315             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
316             {
317                 m_iface = iface;
318             }
319
320             void SetUpdateTime (Time updateTime)
321             {
322                 m_updateTime = updateTime;
323             }
324
325             void SetNextHop (Ipv4Address nextHop)
326             {
327                 m_nextHop = nextHop;
328             }
329
330             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
331             {
332                 m_iface = iface;
333             }
334
335             void SetUpdateTime (Time updateTime)
336             {
337                 m_updateTime = updateTime;
338             }
339
340             void SetNextHop (Ipv4Address nextHop)
341             {
342                 m_nextHop = nextHop;
343             }
344
345             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
346             {
347                 m_iface = iface;
348             }
349
350             void SetUpdateTime (Time updateTime)
351             {
352                 m_updateTime = updateTime;
353             }
354
355             void SetNextHop (Ipv4Address nextHop)
356             {
357                 m_nextHop = nextHop;
358             }
359
360             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
361             {
362                 m_iface = iface;
363             }
364
365             void SetUpdateTime (Time updateTime)
366             {
367                 m_updateTime = updateTime;
368             }
369
370             void SetNextHop (Ipv4Address nextHop)
371             {
372                 m_nextHop = nextHop;
373             }
374
375             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
376             {
377                 m_iface = iface;
378             }
379
380             void SetUpdateTime (Time updateTime)
381             {
382                 m_updateTime = updateTime;
383             }
384
385             void SetNextHop (Ipv4Address nextHop)
386             {
387                 m_nextHop = nextHop;
388             }
389
390             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
391             {
392                 m_iface = iface;
393             }
394
395             void SetUpdateTime (Time updateTime)
396             {
397                 m_updateTime = updateTime;
398             }
399
400             void SetNextHop (Ipv4Address nextHop)
401             {
402                 m_nextHop = nextHop;
403             }
404
405             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
406             {
407                 m_iface = iface;
408             }
409
410             void SetUpdateTime (Time updateTime)
411             {
412                 m_updateTime = updateTime;
413             }
414
415             void SetNextHop (Ipv4Address nextHop)
416             {
417                 m_nextHop = nextHop;
418             }
419
420             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
421             {
422                 m_iface = iface;
423             }
424
425             void SetUpdateTime (Time updateTime)
426             {
427                 m_updateTime = updateTime;
428             }
429
430             void SetNextHop (Ipv4Address nextHop)
431             {
432                 m_nextHop = nextHop;
433             }
434
435             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
436             {
437                 m_iface = iface;
438             }
439
440             void SetUpdateTime (Time updateTime)
441             {
442                 m_updateTime = updateTime;
443             }
444
445             void SetNextHop (Ipv4Address nextHop)
446             {
447                 m_nextHop = nextHop;
448             }
449
450             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
451             {
452                 m_iface = iface;
453             }
454
455             void SetUpdateTime (Time updateTime)
456             {
457                 m_updateTime = updateTime;
458             }
459
460             void SetNextHop (Ipv4Address nextHop)
461             {
462                 m_nextHop = nextHop;
463             }
464
465             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
466             {
467                 m_iface = iface;
468             }
469
470             void SetUpdateTime (Time updateTime)
471             {
472                 m_updateTime = updateTime;
473             }
474
475             void SetNextHop (Ipv4Address nextHop)
476             {
477                 m_nextHop = nextHop;
478             }
479
480             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
481             {
482                 m_iface = iface;
483             }
484
485             void SetUpdateTime (Time updateTime)
486             {
487                 m_updateTime = updateTime;
488             }
489
490             void SetNextHop (Ipv4Address nextHop)
491             {
492                 m_nextHop = nextHop;
493             }
494
495             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
496             {
497                 m_iface = iface;
498             }
499
500             void SetUpdateTime (Time updateTime)
501             {
502                 m_updateTime = updateTime;
503             }
504
505             void SetNextHop (Ipv4Address nextHop)
506             {
507                 m_nextHop = nextHop;
508             }
509
510             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
511             {
512                 m_iface = iface;
513             }
514
515             void SetUpdateTime (Time updateTime)
516             {
517                 m_updateTime = updateTime;
518             }
519
520             void SetNextHop (Ipv4Address nextHop)
521             {
522                 m_nextHop = nextHop;
523             }
524
525             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
526             {
527                 m_iface = iface;
528             }
529
530             void SetUpdateTime (Time updateTime)
531             {
532                 m_updateTime = updateTime;
533             }
534
535             void SetNextHop (Ipv4Address nextHop)
536             {
537                 m_nextHop = nextHop;
538             }
539
540             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
541             {
542                 m_iface = iface;
543             }
544
545             void SetUpdateTime (Time updateTime)
546             {
547                 m_updateTime = updateTime;
548             }
549
550             void SetNextHop (Ipv4Address nextHop)
551             {
552                 m_nextHop = nextHop;
553             }
554
555             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
556             {
557                 m_iface = iface;
558             }
559
560             void SetUpdateTime (Time updateTime)
561             {
562                 m_updateTime = updateTime;
563             }
564
565             void SetNextHop (Ipv4Address nextHop)
566             {
567                 m_nextHop = nextHop;
568             }
569
570             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
571             {
572                 m_iface = iface;
573             }
574
575             void SetUpdateTime (Time updateTime)
576             {
577                 m_updateTime = updateTime;
578             }
579
580             void SetNextHop (Ipv4Address nextHop)
581             {
582                 m_nextHop = nextHop;
583             }
584
585             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
586             {
587                 m_iface = iface;
588             }
589
590             void SetUpdateTime (Time updateTime)
591             {
592                 m_updateTime = updateTime;
593             }
594
595             void SetNextHop (Ipv4Address nextHop)
596             {
597                 m_nextHop = nextHop;
598             }
599
600             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
601             {
602                 m_iface = iface;
603             }
604
605             void SetUpdateTime (Time updateTime)
606             {
607                 m_updateTime = updateTime;
608             }
609
610             void SetNextHop (Ipv4Address nextHop)
611             {
612                 m_nextHop = nextHop;
613             }
614
615             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
616             {
617                 m_iface = iface;
618             }
619
620             void SetUpdateTime (Time updateTime)
621             {
622                 m_updateTime = updateTime;
623             }
624
625             void SetNextHop (Ipv4Address nextHop)
626             {
627                 m_nextHop = nextHop;
628             }
629
630             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
631             {
632                 m_iface = iface;
633             }
634
635             void SetUpdateTime (Time updateTime)
636             {
637                 m_updateTime = updateTime;
638             }
639
640             void SetNextHop (Ipv4Address nextHop)
641             {
642                 m_nextHop = nextHop;
643             }
644
645             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
646             {
647                 m_iface = iface;
648             }
649
650             void SetUpdateTime (Time updateTime)
651             {
652                 m_updateTime = updateTime;
653             }
654
655             void SetNextHop (Ipv4Address nextHop)
656             {
657                 m_nextHop = nextHop;
658             }
659
660             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
661             {
662                 m_iface = iface;
663             }
664
665             void SetUpdateTime (Time updateTime)
666             {
667                 m_updateTime = updateTime;
668             }
669
670             void SetNextHop (Ipv4Address nextHop)
671             {
672                 m_nextHop = nextHop;
673             }
674
675             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
676             {
677                 m_iface = iface;
678             }
679
680             void SetUpdateTime (Time updateTime)
681             {
682                 m_updateTime = updateTime;
683             }
684
685             void SetNextHop (Ipv4Address nextHop)
686             {
687                 m_nextHop = nextHop;
688             }
689
690             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
691             {
692                 m_iface = iface;
693             }
694
695             void SetUpdateTime (Time updateTime)
696             {
697                 m_updateTime = updateTime;
698             }
699
700             void SetNextHop (Ipv4Address nextHop)
701             {
702                 m_nextHop = nextHop;
703             }
704
705             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
706             {
707                 m_iface = iface;
708             }
709
710             void SetUpdateTime (Time updateTime)
711             {
712                 m_updateTime = updateTime;
713             }
714
715             void SetNextHop (Ipv4Address nextHop)
716             {
717                 m_nextHop = nextHop;
718             }
719
720             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
721             {
722                 m_iface = iface;
723             }
724
725             void SetUpdateTime (Time updateTime)
726             {
727                 m_updateTime = updateTime;
728             }
729
730             void SetNextHop (Ipv4Address nextHop)
731             {
732                 m_nextHop = nextHop;
733             }
734
735             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
736             {
737                 m_iface = iface;
738             }
739
740             void SetUpdateTime (Time updateTime)
741             {
742                 m_updateTime = updateTime;
743             }
744
745             void SetNextHop (Ipv4Address nextHop)
746             {
747                 m_nextHop = nextHop;
748             }
749
750             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
751             {
752                 m_iface = iface;
753             }
754
755             void SetUpdateTime (Time updateTime)
756             {
757                 m_updateTime = updateTime;
758             }
759
760             void SetNextHop (Ipv4Address nextHop)
761             {
762                 m_nextHop = nextHop;
763             }
764
765             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
766             {
767                 m_iface = iface;
768             }
769
770             void SetUpdateTime (Time updateTime)
771             {
772                 m_updateTime = updateTime;
773             }
774
775             void SetNextHop (Ipv4Address nextHop)
776             {
777                 m_nextHop = nextHop;
778             }
779
780             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
781             {
782                 m_iface = iface;
783             }
784
785             void SetUpdateTime (Time updateTime)
786             {
787                 m_updateTime = updateTime;
788             }
789
790             void SetNextHop (Ipv4Address nextHop)
791             {
792                 m_nextHop = nextHop;
793             }
794
795             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
796             {
797                 m_iface = iface;
798             }
799
800             void SetUpdateTime (Time updateTime)
801             {
802                 m_updateTime = updateTime;
803             }
804
805             void SetNextHop (Ipv4Address nextHop)
806             {
807                 m_nextHop = nextHop;
808             }
809
810             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
811             {
812                 m_iface = iface;
813             }
814
815             void SetUpdateTime (Time updateTime)
816             {
817                 m_updateTime = updateTime;
818             }
819
820             void SetNextHop (Ipv4Address nextHop)
821             {
822                 m_nextHop = nextHop;
823             }
824
825             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
826             {
827                 m_iface = iface;
828             }
829
830             void SetUpdateTime (Time updateTime)
831             {
832                 m_updateTime = updateTime;
833             }
834
835             void SetNextHop (Ipv4Address nextHop)
836             {
837                 m_nextHop = nextHop;
838             }
839
840             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
841             {
842                 m_iface = iface;
843             }
844
845             void SetUpdateTime (Time updateTime)
846             {
847                 m_updateTime = updateTime;
848             }
849
850             void SetNextHop (Ipv4Address nextHop)
851             {
852                 m_nextHop = nextHop;
853             }
854
855             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
856             {
857                 m_iface = iface;
858             }
859
860             void SetUpdateTime (Time updateTime)
861             {
862                 m_updateTime = updateTime;
863             }
864
865             void SetNextHop (Ipv4Address nextHop)
866             {
867                 m_nextHop = nextHop;
868             }
869
870             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
871             {
872                 m_iface = iface;
873             }
874
875             void SetUpdateTime (Time updateTime)
876             {
877                 m_updateTime = updateTime;
878             }
879
880             void SetNextHop (Ipv4Address nextHop)
881             {
882                 m_nextHop = nextHop;
883             }
884
885             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
886             {
887                 m_iface = iface;
888             }
889
890             void SetUpdateTime (Time updateTime)
891             {
892                 m_updateTime = updateTime;
893             }
894
895             void SetNextHop (Ipv4Address nextHop)
896             {
897                 m_nextHop = nextHop;
898             }
899
900             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
901             {
902                 m_iface = iface;
903             }
904
905             void SetUpdateTime (Time updateTime)
906             {
907                 m_updateTime = updateTime;
908             }
909
910             void SetNextHop (Ipv4Address nextHop)
911             {
912                 m_nextHop = nextHop;
913             }
914
915             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
916             {
917                 m_iface = iface;
918             }
919
920             void SetUpdateTime (Time updateTime)
921             {
922                 m_updateTime = updateTime;
923             }
924
925             void SetNextHop (Ipv4Address nextHop)
926             {
927                 m_nextHop = nextHop;
928             }
929
930             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
931             {
932                 m_iface = iface;
933             }
934
935             void SetUpdateTime (Time updateTime)
936             {
937                 m_updateTime = updateTime;
938             }
939
940             void SetNextHop (Ipv4Address nextHop)
941             {
942                 m_nextHop = nextHop;
943             }
944
945             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
946             {
947                 m_iface = iface;
948             }
949
950             void SetUpdateTime (Time updateTime)
951             {
952                 m_updateTime = updateTime;
953             }
954
955             void SetNextHop (Ipv4Address nextHop)
956             {
957                 m_nextHop = nextHop;
958             }
959
960             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
961             {
962                 m_iface = iface;
963             }
964
965             void SetUpdateTime (Time updateTime)
966             {
967                 m_updateTime = updateTime;
968             }
969
970             void SetNextHop (Ipv4Address nextHop)
971             {
972                 m_nextHop = nextHop;
973             }
974
975             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
976             {
977                 m_iface = iface;
978             }
979
980             void SetUpdateTime (Time updateTime)
981             {
982                 m_updateTime = updateTime;
983             }
984
985             void SetNextHop (Ipv4Address nextHop)
986             {
987                 m_nextHop = nextHop;
988             }
989
990             void SetInterfaceAddress (Ipv4InterfaceAddress iface)
991             {
992                 m_iface = iface;
993             }
994
995             void SetUpdateTime (Time updateTime)
996             {
997                 m_updateTime = updateTime;
998             }
999
1000            void SetNextHop (Ipv4Address nextHop)
1001            {
1002                m_nextHop = nextHop;
1003            }
1004
1005            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1006            {
1007                m_iface = iface;
1008            }
1009
1010            void SetUpdateTime (Time updateTime)
1011            {
1012                m_updateTime = updateTime;
1013            }
1014
1015            void SetNextHop (Ipv4Address nextHop)
1016            {
1017                m_nextHop = nextHop;
1018            }
1019
1020            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1021            {
1022                m_iface = iface;
1023            }
1024
1025            void SetUpdateTime (Time updateTime)
1026            {
1027                m_updateTime = updateTime;
1028            }
1029
1030            void SetNextHop (Ipv4Address nextHop)
1031            {
1032                m_nextHop = nextHop;
1033            }
1034
1035            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1036            {
1037                m_iface = iface;
1038            }
1039
1040            void SetUpdateTime (Time updateTime)
1041            {
1042                m_updateTime = updateTime;
1043            }
1044
1045            void SetNextHop (Ipv4Address nextHop)
1046            {
1047                m_nextHop = nextHop;
1048            }
1049
1050            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1051            {
1052                m_iface = iface;
1053            }
1054
1055            void SetUpdateTime (Time updateTime)
1056            {
1057                m_updateTime = updateTime;
1058            }
1059
1060            void SetNextHop (Ipv4Address nextHop)
1061            {
1062                m_nextHop = nextHop;
1063            }
1064
1065            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1066            {
1067                m_iface = iface;
1068            }
1069
1070            void SetUpdateTime (Time updateTime)
1071            {
1072                m_updateTime = updateTime;
1073            }
1074
1075            void SetNextHop (Ipv4Address nextHop)
1076            {
1077                m_nextHop = nextHop;
1078            }
1079
1080            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1081            {
1082                m_iface = iface;
1083            }
1084
1085            void SetUpdateTime (Time updateTime)
1086            {
1087                m_updateTime = updateTime;
1088            }
1089
1090            void SetNextHop (Ipv4Address nextHop)
1091            {
1092                m_nextHop = nextHop;
1093            }
1094
1095            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1096            {
1097                m_iface = iface;
1098            }
1099
1100            void SetUpdateTime (Time updateTime)
1101            {
1102                m_updateTime = updateTime;
1103            }
1104
1105            void SetNextHop (Ipv4Address nextHop)
1106            {
1107                m_nextHop = nextHop;
1108            }
1109
1110            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1111            {
1112                m_iface = iface;
1113            }
1114
1115            void SetUpdateTime (Time updateTime)
1116            {
1117                m_updateTime = updateTime;
1118            }
1119
1120            void SetNextHop (Ipv4Address nextHop)
1121            {
1122                m_nextHop = nextHop;
1123            }
1124
1125            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1126            {
1127                m_iface = iface;
1128            }
1129
1130            void SetUpdateTime (Time updateTime)
1131            {
1132                m_updateTime = updateTime;
1133            }
1134
1135            void SetNextHop (Ipv4Address nextHop)
1136            {
1137                m_nextHop = nextHop;
1138            }
1139
1140            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1141            {
1142                m_iface = iface;
1143            }
1144
1145            void SetUpdateTime (Time updateTime)
1146            {
1147                m_updateTime = updateTime;
1148            }
1149
1150            void SetNextHop (Ipv4Address nextHop)
1151            {
1152                m_nextHop = nextHop;
1153            }
1154
1155            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1156            {
1157                m_iface = iface;
1158            }
1159
1160            void SetUpdateTime (Time updateTime)
1161            {
1162                m_updateTime = updateTime;
1163            }
1164
1165            void SetNextHop (Ipv4Address nextHop)
1166            {
1167                m_nextHop = nextHop;
1168            }
1169
1170            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1171            {
1172                m_iface = iface;
1173            }
1174
1175            void SetUpdateTime (Time updateTime)
1176            {
1177                m_updateTime = updateTime;
1178            }
1179
1180            void SetNextHop (Ipv4Address nextHop)
1181            {
1182                m_nextHop = nextHop;
1183            }
1184
1185            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1186            {
1187                m_iface = iface;
1188            }
1189
1190            void SetUpdateTime (Time updateTime)
1191            {
1192                m_updateTime = updateTime;
1193            }
1194
1195            void SetNextHop (Ipv4Address nextHop)
1196            {
1197                m_nextHop = nextHop;
1198            }
1199
1200            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1201            {
1202                m_iface = iface;
1203            }
1204
1205            void SetUpdateTime (Time updateTime)
1206            {
1207                m_updateTime = updateTime;
1208            }
1209
1210            void SetNextHop (Ipv4Address nextHop)
1211            {
1212                m_nextHop = nextHop;
1213            }
1214
1215            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1216            {
1217                m_iface = iface;
1218            }
1219
1220            void SetUpdateTime (Time updateTime)
1221            {
1222                m_updateTime = updateTime;
1223            }
1224
1225            void SetNextHop (Ipv4Address nextHop)
1226            {
1227                m_nextHop = nextHop;
1228            }
1229
1230            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1231            {
1232                m_iface = iface;
1233            }
1234
1235            void SetUpdateTime (Time updateTime)
1236            {
1237                m_updateTime = updateTime;
1238            }
1239
1240            void SetNextHop (Ipv4Address nextHop)
1241            {
1242                m_nextHop = nextHop;
1243            }
1244
1245            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1246            {
1247                m_iface = iface;
1248            }
1249
1250            void SetUpdateTime (Time updateTime)
1251            {
1252                m_updateTime = updateTime;
1253            }
1254
1255            void SetNextHop (Ipv4Address nextHop)
1256            {
1257                m_nextHop = nextHop;
1258            }
1259
1260            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1261            {
1262                m_iface = iface;
1263            }
1264
1265            void SetUpdateTime (Time updateTime)
1266            {
1267                m_updateTime = updateTime;
1268            }
1269
1270            void SetNextHop (Ipv4Address nextHop)
1271            {
1272                m_nextHop = nextHop;
1273            }
1274
1275            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1276            {
1277                m_iface = iface;
1278            }
1279
1280            void SetUpdateTime (Time updateTime)
1281            {
1282                m_updateTime = updateTime;
1283            }
1284
1285            void SetNextHop (Ipv4Address nextHop)
1286            {
1287                m_nextHop = nextHop;
1288            }
1289
1290            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1291            {
1292                m_iface = iface;
1293            }
1294
1295            void SetUpdateTime (Time updateTime)
1296            {
1297                m_updateTime = updateTime;
1298            }
1299
1300            void SetNextHop (Ipv4Address nextHop)
1301            {
1302                m_nextHop = nextHop;
1303            }
1304
1305            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1306            {
1307                m_iface = iface;
1308            }
1309
1310            void SetUpdateTime (Time updateTime)
1311            {
1312                m_updateTime = updateTime;
1313            }
1314
1315            void SetNextHop (Ipv4Address nextHop)
1316            {
1317                m_nextHop = nextHop;
1318            }
1319
1320            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1321            {
1322                m_iface = iface;
1323            }
1324
1325            void SetUpdateTime (Time updateTime)
1326            {
1327                m_updateTime = updateTime;
1328            }
1329
1330            void SetNextHop (Ipv4Address nextHop)
1331            {
1332                m_nextHop = nextHop;
1333            }
1334
1335            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1336            {
1337                m_iface = iface;
1338            }
1339
1340            void SetUpdateTime (Time updateTime)
1341            {
1342                m_updateTime = updateTime;
1343            }
1344
1345            void SetNextHop (Ipv4Address nextHop)
1346            {
1347                m_nextHop = nextHop;
1348            }
1349
1350            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1351            {
1352                m_iface = iface;
1353            }
1354
1355            void SetUpdateTime (Time updateTime)
1356            {
1357                m_updateTime = updateTime;
1358            }
1359
1360            void SetNextHop (Ipv4Address nextHop)
1361            {
1362                m_nextHop = nextHop;
1363            }
1364
1365            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1366            {
1367                m_iface = iface;
1368            }
1369
1370            void SetUpdateTime (Time updateTime)
1371            {
1372                m_updateTime = updateTime;
1373            }
1374
1375            void SetNextHop (Ipv4Address nextHop)
1376            {
1377                m_nextHop = nextHop;
1378            }
1379
1380            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1381            {
1382                m_iface = iface;
1383            }
1384
1385            void SetUpdateTime (Time updateTime)
1386            {
1387                m_updateTime = updateTime;
1388            }
1389
1390            void SetNextHop (Ipv4Address nextHop)
1391            {
1392                m_nextHop = nextHop;
1393            }
1394
1395            void SetInterfaceAddress (Ipv4InterfaceAddress iface)
1396            {
1397                m_iface = iface;
1398            }
1399
1400            void SetUpdateTime (Time updateTime)
14
```

```

78     uint16_t GetHopCount () const
79     {
80         return m_hopCount;
81     }
82     void SetPheromone (double pheromone)
83     {
84         m_pheromone = pheromone;
85     }
86     double GetPheromone () const
87     {
88         return m_pheromone;
89     }
90     void SetResidualEnergy (double residualEnergy)
91     {
92         m_residualEnergy = residualEnergy;
93     }
94     double GetResidualEnergy () const
95     {
96         return m_residualEnergy;
97     }
98     void SetRssi (double rssi)
99     {
100         m_rssi = rssi;
101     }
102     double GetRssi () const
103     {
104         return m_rssi;
105     }
106     void SetNextHop (Ipv4Address nextHop)
107     {
108         //m_nextHop = nextHop;
109         m_ipv4Route->SetGateway (nextHop);
110     }
111     Ipv4Address GetNextHop () const
112     {
113         //return m_nextHop;
114         return m_ipv4Route->GetGateway ();
115     }
116     void SetOutputDevice (Ptr<NetDevice> dev)
117     {
118         m_ipv4Route->SetOutputDevice (dev);
119     }
120     Ptr<NetDevice> GetOutputDevice () const
121     {
122         return m_ipv4Route->GetOutputDevice ();
123     }
124     Ipv4InterfaceAddress GetInterface () const
125     {
126         return m_iface;
127     }
128     void SetInterface (Ipv4InterfaceAddress iface)
129     {
130         m_iface = iface;
131     }
132     void SetUpdatedTime (Time lt)
133     {
134         m_updatedTime = lt + Simulator::Now ();
135     }
136     Time GetUpdatedTime () const
137     {
138         return m_updatedTime - Simulator::Now ();
139     }
140
141
142         /**
143         * \brief Compare source address
144         * \return true if equal
145         */
146     bool operator == (Ipv4Address const dst ) const
147     {
148         //return (m_dest == dst );
149         return (m_ipv4Route->GetDestination () == dst);
150     }
151
152     void Print (Ptr < OutputStreamWrapper > stream) const;
153
154 private:
155
156
157

```

```

158
159 //Ipv4Address m_dest;
160
161 Ptr<Ipv4Route> m_ipv4Route;
162
163 uint16_t m_hopCount;
164
165 double m_pheromone;
166
167 std::vector<Ipv4Address> m_precursorList;
168
169 double m_residualEnergy;
170
171 double m_rssi;
172
173 //Ipv4Address m_nextHop;
174
175 Ipv4InterfaceAddress m_iface;
176
177 Time m_updatedTime;
178
179 };
180
181
182 typedef std::map < Ipv4Address, antcommRoutingTableEntry > Neighbor;
183 typedef std::map < Ipv4Address, Neighbor > RTable; // map the above entries with IPv4 address of
    destination node(s)
184
185 /**
186  * \ingroup antcomm
187  * \brief The Information table used by antcomm protocol
188  */
189 class RoutingTable
190 {
191 public:
192
193     /**
194     * Add routing table entry if it doesn't yet exist in information table
195     * \param r information table entry
196     * \return true in success
197     */
198     //bool AddRoute (antcommRoutingTableEntry & it);
199     bool AddRoute (antcommRoutingTableEntry & it);
200
201     /**
202     * Delete Routing table entry with source address source, if it exists.
203     * \param dst source address
204     * \return true on success
205     */
206     bool DeleteRoute (Ipv4Address dst );
207
208     /**
209     * Lookup Routing table entry with source address source
210     * \param dst source address
211     * \param rt entry with source address source, if exists
212     * \return true on success
213     */
214     bool LookupRoute (Ipv4Address dst, Ipv4Address neighbor, antcommRoutingTableEntry & nt);
215     bool LookupRoute2 (Ipv4Address dst, antcommRoutingTableEntry & nt);
216
217     /// Update Routing table
218     bool Update (antcommRoutingTableEntry & nt);
219
220     Ipv4Address NextHopforDst (antcommRoutingTableEntry & nt);
221
222     /// Print routing table
223     void Print (Ptr < OutputStreamWrapper > stream) const;
224
225 private:
226
227     /* m_routes is a map of map
228     std::map < Ipv4Address, std::map < Ipv4Address, antcommRoutingTableEntry > > m_routes;
229     */
230     RTable m_routes;
231 };
232
233 }
234 }
235
236
237 #endif /* antcomm_RTABLE_H */

```

---



---

```

1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2
3  // Include a header file from your module to test.
4  #include "ns3/antree-module.h"
5
6  // An essential include is test.h
7  #include "ns3/test.h"
8
9  // Do not put your test classes in namespace ns3. You may find it useful
10 // to use the using directive to access the ns3 namespace directly
11 using namespace ns3;
12
13 // This is an example TestCase.
14 class AntreeTestCase1 : public TestCase
15 {
16 public:
17     AntreeTestCase1 ();
18     virtual ~AntreeTestCase1 ();
19
20 private:
21     virtual void DoRun (void);
22 };
23
24 // Add some help text to this case to describe what it is intended to test
25 AntreeTestCase1::AntreeTestCase1 ()
26     : TestCase ("Antree_test_case_(does_nothing)")
27 {
28 }
29
30 // This destructor does nothing but we include it as a reminder that
31 // the test case should clean up after itself
32 AntreeTestCase1::~AntreeTestCase1 ()
33 {
34 }
35
36 //
37 // This method is the pure virtual method from class TestCase that every
38 // TestCase must implement
39 //
40 void
41 AntreeTestCase1::DoRun (void)
42 {
43     // A wide variety of test macros are available in src/core/test.h
44     NS_TEST_ASSERT_MSG_EQ (true, true, "true_doesn't_equal_true_for_some_reason");
45     // Use this one for floating point comparisons
46     NS_TEST_ASSERT_MSG_EQ_TOL (0.01, 0.01, 0.001, "Numbers_are_not_equal_within_tolerance");
47 }
48
49 // The TestSuite class names the TestSuite, identifies what type of TestSuite,
50 // and enables the TestCases to be run. Typically, only the constructor for
51 // this class must be defined
52 //
53 class AntreeTestSuite : public TestSuite
54 {
55 public:
56     AntreeTestSuite ();
57 };
58
59 AntreeTestSuite::AntreeTestSuite ()
60     : TestSuite ("antree", UNIT)
61 {
62     // TestDuration for TestCase can be QUICK, EXTENSIVE or TAKES_FOREVER
63     AddTestCase (new AntreeTestCase1, TestCase::QUICK);
64 }
65
66 // Do not forget to allocate an instance of this TestSuite
67 static AntreeTestSuite antreeTestSuite;

```

---

Table A.1.: Placement of Nodes

Position X	Position Y
0	-2
55	-2
109	-1
150	-7
214	5
259	-6
302	-5
374	1
407	9
428	-2
18	30
67	48
123	44
184	47
231	43
275	58
318	45
384	43
422	54
465	41
-1	84
41	108
112	94
150	90
206	97

*continued on next page*

Table A.1.: *continued*

Position X	Position Y
239	111
283	100
352	103
379	93
446	91
29	154
87	164
125	154
174	154
219	156
265	162
330	155
379	160
421	147
472	156
1	185
64	207
101	192
148	200
187	190
259	208
318	217
355	186
381	180
453	203

*continued on next page*

Table A.1.: *continued*

Position X	Position Y
16	248
66	250
126	257
173	263
222	242
284	255
318	242
377	256
435	262
475	262
-1	283
57	289
80	297
142	307
206	295
231	296
290	302
358	300
416	296
448	292
23	337
69	362
117	350
177	350
220	335

*continued on next page*

Table A.1.: *continued*

Position X	Position Y
282	349
338	339
364	343
420	349
479	345
-3	399
41	414
94	382
157	399
211	400
250	416
302	413
348	386
384	412
464	400
21	441
74	445
135	437
183	463
236	445
282	463
333	437
386	450
427	438
465	438

Table A.2.  
Data of Throughput against background traffic in Ant Colony

Background Traffic	Throughput
0	97478
57.8	97462
126.8	93082.8
211.7	86924.2
310.5	84732.9
465.7	77369
623	73137.4
775.2	70158.4
948.2	66692.6
1118.2	64796.4
1317.4	62058.7

Table A.3.  
Data of Mean Delay against background traffic in Ant Colony

Background Traffic	Mean Delay
0	0.0406364
57.8	0.0455162
126.8	0.0544133
211.7	0.0548401
310.5	0.0592297
465.7	0.0607113
623	0.0618095
775.2	0.0649593
948.2	0.0659632
1118.2	0.0684697
1317.4	0.0686697

Table A.4.  
Data of Packet Loss against background traffic in Ant Colony

Background Traffic	Packet Loss
0	2
57.8	31
126.8	106
211.7	189
310.5	370
465.7	564
623	747
775.2	969
948.2	1170
1118.2	1340
1317.4	1432

Table A.5.  
Throughput vs Average Degree in Ant Colony

Throughput vs Average Degree			
Average Degree	Load (90 kbps)	Load (400 kbps)	Load (725 kbps)
4.94	97760.9	92220.2	92692.4
5.34	97733.9	81543	77623.1
5.94	97623.6	95599.7	95300.2
6.94	97386.9	93169.4	92840
8.03	91075.3	78756.8	74819.3
9.42	91085.5	63038.6	54744.2
10.54	95355.5	84516.5	66927.4
11.63	96353.4	87651.3	86194.6
13.43	60617.5	44129.5	34122.2



Table A.6.  
Mean Delay vs Average Degree in Ant Colony

Mean Delay vs Average Degree			
Average Degree	Load (90 kbps)	Load (400 kbps)	Load (725 kbps)
4.94	0.034309	0.0432472	0.0437581
5.34	0.0463942	0.058193	0.063629
5.94	0.0360623	0.0403674	0.033113
6.94	0.0359187	0.042346	0.0431649
8.03	0.0447374	0.0619078	0.064226
9.42	0.0386595	0.0713462	0.0808875
10.54	0.0372176	0.0480497	0.0540632
11.63	0.0398859	0.0487399	0.0523276
13.43	0.0525717	0.0789278	0.0839992

Table A.7.  
Packet Loss vs Average Degree in Ant Colony

Packet loss vs Average Degree			
Average Degree	Load (90 kbps)	Load (400 kbps)	Load (725 kbps)
4.94	1	133	204
5.34	1	377	803
5.94	1	52	100
6.94	3	106	201
8.03	42	442	942
9.42	46	801	1696
10.54	16	304	1234
11.63	10	237	452
13.43	265	1251	2662

Table A.8.  
Throughput comparison between AODV and Ant Colony

Comparison of Throughput performance		
Background Load (kbps)	Throughput in AODV (kbps)	Throughput in Ant Colony (kbps)
0	97064.8	97478
57.8	97439	97462
126.8	97332.2	93082.8
211.7	96481	86924.2
310.5	94474.925	84732.9
465.7	93643.6	77369
623	89244.675	73137.4
775.2	87875.75	70158.4
948.2	85372.475	66692.6
1118.8	82064.225	64796.4
1317.4	77739.2	62058.7

Table A.9.  
Delay Comparison between AODV and Ant Colony

Comparison of Delay performance		
Background Load (kbps)	Delay in AODV (sec)	Delay in Ant Colony (sec)
0	0.03429795	0.0406364
57.8	0.03234435	0.0455162
126.8	0.0328218	0.0544133
211.7	0.0330905	0.0548401
310.5	0.032899725	0.0592297
465.7	0.034572075	0.0607113
623	0.034309825	0.0618095
775.2	0.03908005	0.0649593
948.2	0.04073455	0.0659632
1118.8	0.05084815	0.0684697
1317.4	0.0565708	0.0686697

Table A.10.  
Packet Loss Comparison between AODV and Ant Colony

Comparison of Packet Loss performance		
Background Load (kbps)	Packet Loss in AODV	Packet Loss in Ant Colony
0	3.5	2
57.8	3.5	31
126.8	4.75	106
211.7	30	189
310.5	81	370
465.7	196	564
623	275	747
775.2	392	969
948.2	558	1170
1118.8	618	1340
1317.4	772.25	1432

Table A.11.  
Throughput vs Average Degree between AODV and Ant Colony

Comparison of Throughput against Average Degree		
Average Degree	Throughput in AODV	Throughput in Ant Colony
4.94	97170.9	92692.4
5.34	96532.3	77623.1
5.94	93259.6	95300.2
6.94	82998.6	92840
8.03	67433	74819.3
9.42	71188	54744.2
10.54	31164	66927.4
11.63	26226	86194.6

Table A.12.  
Packet Loss vs Average Degree between AODV and Ant Colony

Comparison of Packet Loss against Average Degree		
Average Degree	Packet Loss in AODV	Packet Loss in Ant Colony
4.94	26	204
5.34	52	803
5.94	185	100
6.94	602	201
8.03	1215	942
9.42	1081	1696
10.54	3000	1234
11.63	3274	452

Table A.13.  
Delay vs Average Degree between AODV and Ant Colony

Comparison of Delay against Average Degree		
Average Degree	Delay in AODV	Delay in Ant Colony
4.94	0.0325634	0.0437581
5.34	0.0302164	0.063629
5.94	0.027723	0.033113
6.94	0.053475	0.0431649
8.03	0.0347015	0.064226
9.42	0.0486191	0.0808875
10.54	0.0586493	0.0540632
11.63	0.0276077	0.0523276

Table A.14.  
Std. Dev. of Residual Energy and Avg Residual Energy in AODV

Time (sec)	Std. Dev of Remaining Energy	Average Residual Energy
50	0.004362	0.360862
75	0.006586	0.342027
100	0.008627	0.32312
125	0.01075	0.304148
150	0.012362	0.285476
175	0.014043	0.266907
200	0.015738	0.24848
225	0.01761	0.22992
250	0.019177	0.211082
275	0.02043	0.192541
300	0.021627	0.173883
325	0.022866	0.15545
350	0.024275	0.136809
375	0.025676	0.118054
400	0.027143	0.099187
425	0.028562	0.080547
450	0.029961	0.061782



Table A.15.  
Std. Dev. of Residual Energy and Avg Residual Energy in Ant Colony

Time (sec)	Std. Dev of Remaining Energy	Avg Residual Energy
50	0.000767	0.368974
75	0.000797	0.354565
100	0.000862	0.340102
125	0.00093	0.325639
150	0.000979	0.311195
175	0.001049	0.296738
200	0.0011	0.282297
225	0.001152	0.267879
250	0.001212	0.253415
275	0.001253	0.238983
300	0.001299	0.224578
325	0.001363	0.210151
350	0.00138	0.195768
375	0.001467	0.181294
400	0.001534	0.166846
425	0.001607	0.152396
450	0.001641	0.137967

Table A.16.  
Hop Comparison between AODV and Ant Colony when Avg Degree is 4.94

Time (sec)	Hop in AODV	Hop in Ant Colony
25	14	14.86667
50	14	15.73333
75	14	14.53333
100	14	15.36667
125	14	15.5
150	14	15.36667
175	14	15.43333
200	14	14.6
225	14	15.26667
250	14	15.63333
275	14	14.9
300	14	16.03333
325	14	15.06667
350	14	15.7
375	14	14.7
400	14	15.63333
425	14	15.8
450	14	15.6
475	14	15.83333

Table A.17.  
Hop Comparison between AODV and Ant Colony when Avg Degree is 5.94

Time (sec)	Hop in AODV	Hop in Ant Colony
25	11	17
50	11	15.66667
75	11	15.16667
100	11	16.03333
125	11	16.63333
150	11	15.86667
175	11	15.7
200	11	16.06667
225	11	15.6
250	11	15.33333
275	11	16.3
300	11	16.9
325	11	15.7
350	11	14.96667
375	11	16.73333
400	11	16.1
425	11	15.36667
450	11	15.2
475	11	15.1

Table A.18.: Remaining Energy of Nodes over different time stamps using Ant Colony

Node ID	T=100s	T=200s	T=300s	T=400s
0	0.341465	0.284081	0.226477	0.169066
1	0.340104	0.28191	0.22384	0.165946
2	0.340241	0.282562	0.224755	0.167184
3	0.340873	0.283225	0.225619	0.168111
4	0.339973	0.282359	0.22487	0.167369
5	0.340057	0.282507	0.224917	0.167159
6	0.341423	0.28399	0.226531	0.169262
7	0.341114	0.28355	0.225958	0.168582
8	0.340822	0.283156	0.225435	0.167847
9	0.339961	0.282386	0.224533	0.166622
10	0.340286	0.282317	0.224391	0.166539
11	0.340328	0.282636	0.22484	0.16727
12	0.338346	0.280131	0.221904	0.163669
13	0.338205	0.280205	0.222377	0.164378
14	0.338564	0.280817	0.222971	0.164953
15	0.340903	0.283007	0.225474	0.16815
16	0.341319	0.284071	0.226718	0.169274
17	0.339768	0.282104	0.224454	0.166714
18	0.33886	0.28076	0.222448	0.164438
19	0.34052	0.282796	0.225126	0.167599
20	0.341866	0.284254	0.226557	0.169001
21	0.341668	0.284423	0.227091	0.169741
22	0.340073	0.282227	0.224503	0.166746
23	0.339011	0.281157	0.22339	0.165659

*continued on next page*

Table A.18.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
24	0.339047	0.281239	0.22361	0.1658
25	0.339607	0.281412	0.22366	0.165997
26	0.341481	0.284313	0.227138	0.169855
27	0.34101	0.28346	0.225989	0.168676
28	0.340883	0.283527	0.225762	0.16809
29	0.340634	0.282945	0.225584	0.167831
30	0.340995	0.283111	0.22525	0.167542
31	0.339797	0.282075	0.224153	0.16641
32	0.33915	0.281355	0.223407	0.165405
33	0.339761	0.281694	0.223893	0.165818
34	0.33874	0.28081	0.222853	0.164829
35	0.340663	0.283041	0.225442	0.167796
36	0.340289	0.282482	0.224532	0.166912
37	0.339865	0.281701	0.223886	0.166616
38	0.339756	0.281715	0.223553	0.165703
39	0.341443	0.283802	0.226187	0.169
40	0.3421	0.284995	0.2278	0.170696
41	0.340259	0.282549	0.225119	0.167581
42	0.33995	0.282233	0.224613	0.166835
43	0.338519	0.280264	0.221804	0.163601
44	0.33931	0.281365	0.223488	0.165697
45	0.339943	0.282059	0.224144	0.16655
46	0.34042	0.282595	0.224841	0.16722
47	0.339655	0.281565	0.223491	0.165925
48	0.340484	0.282323	0.224342	0.16686

*continued on next page*

Table A.18.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
49	0.340253	0.282865	0.225655	0.167997
50	0.340409	0.282279	0.224639	0.167264
51	0.339282	0.281417	0.222721	0.165089
52	0.339174	0.281162	0.223744	0.165791
53	0.339956	0.281995	0.224009	0.166335
54	0.339224	0.281203	0.223191	0.164998
55	0.339418	0.281344	0.223595	0.165704
56	0.340576	0.282519	0.224819	0.167443
57	0.339683	0.281892	0.224323	0.166649
58	0.33883	0.280542	0.222594	0.164952
59	0.340431	0.282978	0.225458	0.168083
60	0.340429	0.282974	0.224982	0.167451
61	0.338996	0.280482	0.222542	0.164647
62	0.339826	0.28166	0.223542	0.165861
63	0.339625	0.281781	0.223928	0.166012
64	0.339233	0.281369	0.22372	0.16597
65	0.339594	0.281601	0.224028	0.166378
66	0.340425	0.282518	0.22445	0.166802
67	0.33952	0.281214	0.223188	0.165386
68	0.339325	0.281044	0.223402	0.165686
69	0.339632	0.282045	0.224571	0.166828
70	0.340365	0.282424	0.224782	0.167254
71	0.339509	0.281754	0.224265	0.166247
72	0.339972	0.282104	0.224624	0.166779
73	0.338836	0.280996	0.223185	0.165266

*continued on next page*

Table A.18.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
74	0.34104	0.283742	0.22637	0.168972
75	0.341018	0.283456	0.225973	0.168414
76	0.339781	0.281263	0.2233	0.165361
77	0.340696	0.282996	0.225085	0.167349
78	0.340678	0.282913	0.225334	0.167645
79	0.34103	0.283566	0.226195	0.168767
80	0.340725	0.283459	0.226274	0.168762
81	0.33877	0.280468	0.222561	0.164212
82	0.340178	0.282609	0.225364	0.167481
83	0.340926	0.283416	0.226041	0.16874
84	0.339943	0.282216	0.224653	0.167131
85	0.339803	0.281657	0.224145	0.166126
86	0.33945	0.281364	0.223665	0.16561
87	0.338659	0.280441	0.222078	0.163336
88	0.340223	0.28245	0.224535	0.166115
89	0.341613	0.284182	0.226766	0.169167
90	0.340371	0.282589	0.224947	0.166833
91	0.340022	0.282514	0.225245	0.167676
92	0.340522	0.28275	0.22489	0.16681
93	0.341423	0.283892	0.226625	0.169326
94	0.339687	0.281557	0.223961	0.16585
95	0.339413	0.281298	0.223589	0.165149
96	0.33921	0.280967	0.223022	0.164273
97	0.340496	0.283033	0.225082	0.167014
98	0.340525	0.282828	0.225057	0.167008

*continued on next page*

Table A.18.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
99	0.341924	0.28465	0.22737	0.170042

Table A.19.: Remaining Energy of Nodes over different time stamps using AODV

Node ID	T=100s	T=200s	T=300s	T=400s
0	0.332787	0.267067	0.202308	0.13745
1	0.328895	0.258227	0.188063	0.119458
2	0.335387	0.271528	0.206096	0.138221
3	0.332696	0.267222	0.200556	0.132197
4	0.323235	0.250811	0.177245	0.106459
5	0.322828	0.248777	0.175692	0.10443
6	0.335273	0.270925	0.204009	0.138158
7	0.334938	0.271518	0.203592	0.136369
8	0.331367	0.264843	0.192755	0.122216
9	0.328967	0.261578	0.187344	0.114485
10	0.328988	0.256726	0.183339	0.111387
11	0.329349	0.262496	0.194921	0.125296
12	0.319249	0.24041	0.161026	0.081824
13	0.318357	0.243126	0.166417	0.089973
14	0.326326	0.249293	0.165885	0.087848
15	0.324294	0.25799	0.190004	0.116952
16	0.335287	0.26851	0.198843	0.131318
17	0.323034	0.252161	0.176158	0.102115
18	0.31472	0.24066	0.161071	0.081001

*continued on next page*



Table A.19.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
19	0.327221	0.259713	0.188593	0.117693
20	0.334562	0.271483	0.207173	0.142014
21	0.339308	0.280295	0.218552	0.155836
22	0.329139	0.261511	0.191406	0.118936
23	0.327867	0.25947	0.187467	0.113637
24	0.310109	0.233083	0.155431	0.076186
25	0.321971	0.25085	0.17468	0.095426
26	0.330063	0.263083	0.192583	0.125098
27	0.323766	0.25277	0.183785	0.11618
28	0.328666	0.25909	0.187244	0.115389
29	0.327279	0.260267	0.191575	0.121981
30	0.324258	0.258334	0.18888	0.115756
31	0.32017	0.24355	0.168074	0.090938
32	0.307693	0.223347	0.140676	0.058419
33	0.316195	0.237827	0.156137	0.077571
34	0.318168	0.24142	0.15646	0.071226
35	0.317989	0.244496	0.169953	0.094536
36	0.307359	0.22082	0.13528	0.051603
37	0.307679	0.22133	0.137046	0.055047
38	0.309944	0.229927	0.15032	0.071286
39	0.325798	0.256345	0.189285	0.122804
40	0.340492	0.282147	0.223476	0.163903
41	0.321159	0.250845	0.178363	0.102225
42	0.31524	0.235316	0.156051	0.075354
43	0.309937	0.225888	0.139425	0.052846

*continued on next page*

Table A.19.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
44	0.316945	0.240377	0.159744	0.081195
45	0.32303	0.242662	0.161019	0.079047
46	0.313458	0.226555	0.142267	0.055014
47	0.309206	0.223791	0.142352	0.062921
48	0.307914	0.22082	0.135937	0.054305
49	0.319558	0.244409	0.170378	0.098469
50	0.332528	0.261635	0.187719	0.110597
51	0.319025	0.235728	0.157836	0.076375
52	0.312214	0.233849	0.155344	0.073312
53	0.324319	0.249167	0.174497	0.099221
54	0.321115	0.242626	0.162238	0.075967
55	0.313709	0.234149	0.155883	0.072326
56	0.320479	0.239469	0.161173	0.081739
57	0.31257	0.225374	0.141589	0.062089
58	0.309686	0.221738	0.13854	0.058453
59	0.321028	0.24583	0.172622	0.10034
60	0.333644	0.261181	0.188415	0.111091
61	0.325104	0.248164	0.168314	0.080323
62	0.325105	0.251012	0.175786	0.098182
63	0.318985	0.238915	0.159059	0.078646
64	0.317792	0.240924	0.162386	0.080022
65	0.323731	0.251241	0.179038	0.104283
66	0.318005	0.239739	0.16003	0.076011
67	0.315395	0.226633	0.141791	0.059918
68	0.307857	0.21822	0.132903	0.048526

*continued on next page*

Table A.19.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
69	0.320143	0.237787	0.159859	0.086817
70	0.33328	0.260481	0.187245	0.111736
71	0.331321	0.2552	0.179956	0.105992
72	0.323267	0.248646	0.174626	0.102011
73	0.314772	0.231146	0.150624	0.070472
74	0.32253	0.252828	0.183058	0.11091
75	0.32927	0.262237	0.196232	0.127453
76	0.303814	0.217649	0.13821	0.058535
77	0.308128	0.223441	0.142832	0.063998
78	0.327082	0.249309	0.17596	0.1071
79	0.324916	0.252619	0.180782	0.108547
80	0.333629	0.269467	0.205296	0.13895
81	0.326967	0.251589	0.175555	0.099256
82	0.334601	0.262149	0.192008	0.125222
83	0.327566	0.257585	0.186516	0.114563
84	0.325324	0.249858	0.175507	0.099204
85	0.323293	0.240744	0.163658	0.088853
86	0.317425	0.232421	0.153416	0.078733
87	0.308453	0.215455	0.133234	0.055484
88	0.318979	0.241224	0.16552	0.094875
89	0.333471	0.26599	0.201629	0.138348
90	0.330645	0.261805	0.19237	0.120844
91	0.331974	0.265291	0.197226	0.126145
92	0.331236	0.259782	0.189974	0.118529
93	0.335239	0.271529	0.20481	0.137616

*continued on next page*

Table A.19.: *continued*

Node ID	T=100s	T=200s	T=300s	T=400s
94	0.330434	0.253798	0.179548	0.103815
95	0.320832	0.236944	0.158846	0.084013
96	0.315952	0.231257	0.151792	0.07664
97	0.329066	0.258353	0.190882	0.126378
98	0.33121	0.26496	0.197831	0.12957
99	0.336774	0.275134	0.213194	0.150685